TEC-0064

RADIUS: Model-Based Optimization

Thomas M. Strat
Pascal V. Fua
Lynn H. Quam

SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025-3493

July 1995

US Army Corps of Engineers
Topographic Engineering Center

T E C

Prepared for:
Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington, VA 22203-1714

Monitored by:
U.S. Army Corps of Engineers
Topographic Engineering Center
7701 Telegraph Road
Alexandria, Virginia 22315-3864

Destroy this report when no longer needed.
Do not return it to the originator.

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

The citation in this report of trade names of commercially available products does not constitute official endorsement or approval of the use of such products.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 1995	Second Annual Oct. 1994 - Apr. 1995

4. TITLE AND SUBTITLE

RADIUS: Model-Based Optimization

5. FUNDING NUMBERS

DACA76-92-C-0034

6. AUTHOR(S)

Thomas M. Strat Lynn H. Quam
Pascal V. Fua

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025-3493

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Advanced Research Projects Agency
3701 North Fairfax Drive, Arlington, VA 22203-1714

U.S. Army Topographic Engineering Center
7701 Telegraph Road., Alexandria, VA 22315-3864

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

TEC-0064

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The construction and use of 3D models of military and industrial sites will allow revolutionary advances in the speed, confidence, and range of analytical techniques with which an Image Analyst (IA) develops and reports intelligence information. This SRI research project, in support of the RADIUS Program, seeks to increase the speed and accuracy with which site models can be constructed from current imagery by developing a new family of image understanding (IU) techniques, and by developing a novel way for an IA to employ them.

14. SUBJECT TERMS

Computer vision, aerial image analysis, RADIUS, optimization, snakes

15. NUMBER OF PAGES

71

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

Contents

List of Figures

List of Tables

PREFACE

This research is sponsored by the Advanced Research Projects Agency (ARPA) and monitored by the U.S. Army Topographic Engineering Center (TEC) under Contract DACA76-92-C-0034, titled, "RADIUS: Model-Based Optimization (Second Annual Report)." The ARPA Program Manager is Dr. Oscar Firschein, and the TEC Contracting Officer's Representative is Ms. Lauretta Williams.

1 Introduction

The construction and use of 3D models of military and industrial sites will allow revolutionary advances in the speed, confidence, and range of analytical techniques with which an Image Analyst (IA) develops and reports intelligence information. This SRI research project in support of the RADIUS Program seeks to increase the speed and accuracy with which site models can be constructed from current imagery by developing a new family of image understanding (IU) techniques, and by developing a novel way for an IA to employ them.

The research is proceeding on two fronts simultaneously:

- Extending the Model-Based Optimization algorithms; and
- Developing a context-based approach to image analyst control of IU algorithms.

This report describes activities during the past 12 months directed toward advancing the state-of-the-art in these two areas.

2 Research Goals

Model-Supported Exploitation (MSE) is the analysis of imagery (by human or computer) with the aid of 2D and 3D models of the scene [1, 2, 3]. There are two major scientific problems that must be solved for MSE to be a viable concept for use in an operational intelligence setting. A successful MSE system must have:

- an interface that enables the image analyst (IA) to easily specify what he wants the machine to do; and
- a set of algorithms that enables the machine to perform the tasks posed by the IA.

While the majority of research in IU has been concerned with fully automated algorithms for interpreting images, the Perception Group at SRI has made the design and implementation of semiautomated systems a major goal. As a result, SRI has made significant progress toward achieving the goals listed above.

- The RADIUS Common Development Environment (RCDE), which is based on the SRI Cartographic Modeling Environment (CME), contains many innovative mechanisms for allowing the user to visualize scene geometry and to control the invocation of site modeling tools. For example, when a user points to something in an image, the ambiguity of his gesture is resolved through the use of a narrowly constrained geometric context. The RCDE is the framework to which we have added new constructs that have been designed especially for model-supported exploitation [4, 5].
- Model-Based Optimization (MBO) algorithms, in which an objective function is optimized to determine the best fit with image data, provide an ideal basis for semiautomated site modeling. The so-called "snake" technology is a particular example of model-based optimization, named for the way the curves wiggle during optimization [6, 7]. SRI has designed and implemented numerous techniques for finding such objects as roads, buildings, vegetation, and rivers. The ability to fit a deformable object model to multiple images

simultaneously, and the employment of continuation methods to avoid local minima, are examples of our advances [8, 9, 10].

SRI is extending the MBO technology to provide the capability of extracting many different object classes under a wide variety of imaging and scene conditions. When integrated into the RCDE, this technology will constitute an operational language tailored to the needs of the intelligence analyst.

3 Model-Based Optimization

Model-Based Optimization (MBO) is a paradigm in which an objective function is used to express both geometric and photometric constraints on features of interest. A model of a feature (such as a road, a building, or coastline) is extracted from an image by adjusting the model until a minimum value of the objective function is obtained. The optimization procedure yields a description that simultaneously satisfies (or nearly satisfies) all constraints, and, as a result, is likely to be a good model of the feature.

Implementation of a Model-Based Optimization algorithm requires the specification of four components.

Objective function: A mathematic function that expresses the preferred geometric and photometric properties to be exhibited by the feature.

Topology: The geometric primitive used to represent the feature, which thereby limits the class of features that can be modeled.

Optimization: The procedure to be employed for finding a configuration of the feature that minimizes (locally) the objective function.

Initial conditions: The configuration of the feature to be used as the starting point by the optimization procedure.

Our research is addressing all four of these areas, in seeking to find instantiations of the MBO paradigm that provide effective means for extracting features of interest to the RADIUS Program.

The applicability of MBO is currently limited by the expressive power of terms in the objective function and by the difficulty of optimization. This project is extending the range of objects that can be modeled within the MBO paradigm, and is developing suitable optimization procedures to support them.

With these extensions, the MBO technology can be used:

1. interactively, to extract objects that are of special interest or that were missed by a fully automated system, or

2. automatically, to extract from an image all objects of a given type.

During the first year of this project, we implemented MBO algorithms for extracting roads, railroads, and other linear features from overhead imagery. Evaluations performed with operational imagery have shown that extraction of such features using MBO reduces the number

of vertices that must be specified to one-third of those required by manual extraction. Further improvement in the optimization procedures is needed to increase the range of features that can be extracted.

3.1 Snakes

Snakes were originated by Terzopoulos, Witkin, and Kass [6, 7] and have since given rise to a large body of literature. Further research by Fua and Leclerc has extended the theory and increased its practicality [11].

A 2–D snake is treated as a polygonal curve $\mathcal{C}$ defined by a set S containing n equidistant vertices

$$S = \{(x_i\ y_i),\ i = 1, \ldots, n\}$$

that can deform itself to optimize an objective function $\mathcal{E}(\mathcal{C})$.

Formally we can write the energy $\mathcal{E}(\mathcal{C})$ that the snake minimizes as a weighted sum of the form

$$\mathcal{E}(\mathcal{C}) = \sum_i \lambda_i \mathcal{E}_i(\mathcal{C}) \tag{1}$$

where the magnitudes of the $\mathcal{E}_i$ depend on the specific radiometry and geometry of the particular scene under consideration and are not necessarily commensurate. In order to determine the values of the λ_i weights in a context-specific way as opposed to an image-specific one, we have found it necessary to normalize out those influences. The dynamics of the optimization are controlled by the gradient of the objective function. We have therefore found that an effective way to achieve this result is to specify a set of normalized weights λ_i' such that

$$\sum_{1 \le i \le n} \lambda_i' = 1\ .$$

The λ_i' define the relative influences of the various components, and we use them to compute the λ_i as follows:

$$\lambda_i = \frac{\lambda_i'}{\| \overrightarrow{\nabla} \mathcal{E}_i(\mathcal{S}^0) \|}$$

where $\mathcal{S}^0$ is the estimate at the start of each optimization step. In this way we ensure that the contribution of each $\mathcal{E}_i$ term is roughly proportional to the corresponding λ_i' independent of the specific image or curve being considered.

3.2 Objective Function

The objective function is used to represent the constraints that an MBO primitive is expected to obey. Each term $\mathcal{E}_i(\mathcal{C})$ in the objective function (Equation refeq:sum) encodes a particular geometric or photometric constraint. Some of the more commonly used terms are listed in Table 1. A precise description of the simplest form of snake is described in Section 3.2.1.

Table 1: Terms used in the objective function to define various snake species

Gene name	Gene value	Description
Photometry	Edge	Prefers to align itself with step edges in the intensity image
	Line	Prefers to align itself with thin lines (dark or light)
Width	Thin	An infinitessimally thin curve
	Fixed-width	A pair of parallel curves separated by a fixed distance
	Variable-width	A pair of approximately parallel curves whose separation can vary at each vertex
Closure	Open	An arbitrary curve represented by a sequence of vertices
	Closed	The boundary of a region denoted by a sequence of vertices in which the first and last vertex are identical
Curvature	Smooth-curve	A sequence of closely spaced vertices with restrictions on allowable smoothness
	Polygonal	A sequence of vertices connected by straight lines with no smoothness constraint
	Rectilinear	A sequence of vertices connected by straight lines that intersect at right angles (in 3–D)
Dimensionality	2–D	A sequence of 2–D vertices in the image plane
	3–D	A sequence of 3–D vertices whose projection in one or more images satisfies the photometric constraints

3.2.1 2–D Linear Snakes

A 2–D snake is treated as a polygonal curve $\mathcal{C}$ defined by a set S containing n equidistant vertices

$$S = \{(x_i\ y_i),\ i = 1, \ldots, n\} \tag{2}$$

that can deform itself to maximize the average edge strength along the curve $\mathcal{G}(\mathcal{C})$:

$$\mathcal{G}(\mathcal{C}) = \frac{1}{|\mathcal{C}|} \int_0^{|\mathcal{C}|} |\nabla \mathcal{I}(\mathbf{f}(s))|\ ds, \tag{3}$$

where I represents the image gray levels, s is the arc length of $\mathcal{C}$, $\mathbf{f}(s)$ is a vector function mapping the arc length s to points (x, y) in the image, and $|\mathcal{C}|$ is the length of $\mathcal{C}$. In practice, $\mathcal{G}(\mathcal{C})$ is computed by sampling the polygonal segments of the curve at regular intervals, looking up the gradient values $|\nabla \mathcal{I}(\mathbf{f}(s))|$ in precomputed gradient images, and summing them up. The gradient images are computed by gaussian smoothing the original image and taking the x and y derivatives to be finite differences of neighboring pixels. It has been shown [11] that the points along a curve that maximizes $\mathcal{G}(\mathcal{C})$ are maxima of the gradient in the direction normal to the curve wherever the curvature is small. Therefore, such a curve approximates edges well except at corners. Unfortunately, $\mathcal{G}(\mathcal{C})$ is not convex functional and to perform the optimization, following Terzopoulos *et al.*, we minimize an energy $\mathcal{E}_(\mathcal{C})$ that is a weighted difference of a regularization term $\mathcal{E}_D(\mathcal{C})$ and of $\mathcal{G}(\mathcal{C})$:

$$\mathcal{E}_(\mathcal{C}) = \lambda_D \mathcal{E}_D(\mathcal{C}) - \lambda_G \mathcal{G}(\mathcal{C}) \tag{4}$$

$$\begin{aligned} \mathcal{E}_D(\mathcal{C}) &= \mu_1 \sum_i (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 \\ &+ \mu_2 \sum_i (2x_i - x_{i-1} - x_{i+1})^2 + (2y_i - y_{i-1} - y_{i+1})^2 \end{aligned} \tag{5}$$

The first term of $\mathcal{E}_D$ approximates the curve's tension and the second term approximates the sum of the square of the curvatures, assuming that the vertices are roughly equidistant. In addition, when starting, as we do, with regularly spaced vertices, this second term tends to maintain that regularity. To perform the optimization we could use the steepest or conjugate gradient, but it would be slow for curves with large numbers of vertices. Instead, it has proven much more effective to embed the curve in a viscous medium and solve the equation of the dynamics

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial S} + \alpha \frac{dS}{dt} &= 0, \\ \text{with } \frac{\partial \mathcal{E}}{\partial S} &= \frac{\partial \mathcal{E}_D}{\partial S} - \frac{\partial \mathcal{G}}{\partial S}, \end{aligned} \tag{6}$$

where $\mathcal{E}$ is the energy of Equation 4, α the viscosity of the medium, and S the state vector of Equation 2 that defines the current position of the curve. Since the deformation energy $\mathcal{E}_D$ in Equation 5 is quadratic, its derivative with respect to S is linear and therefore Equation 6 can be rewritten as

$$\begin{aligned} K_S S_t + \alpha(S_t - S_{t-1}) &= -\left.\frac{\partial \mathcal{E}}{\partial S}\right|_{S_{t-1}} \\ \Rightarrow (K_S + \alpha I) S_t &= \alpha S_{t-1} - \left.\frac{\partial \mathcal{E}}{\partial S}\right|_{S_{t-1}} \end{aligned} \tag{7}$$

where

$$\frac{\partial \mathcal{E}_D}{\partial S} = K_S S,$$

and K_S is a sparse matrix. Note that the derivatives of $\mathcal{E}_D$ with respect to x and y are decoupled so that we can rewrite Equation 7 as a set of two differential equations in the two spatial coordinates

$$\begin{aligned}(K + \alpha I)X_t &= \alpha X_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial X}\right|_{X_{t-1}} \\ (K + \alpha I)Y_t &= \alpha Y_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial Y}\right|_{Y_{t-1}}\end{aligned}$$

where K is a pentadiagonal matrix, and X and Y are the vectors of the x and y vertex coordinates. Because K is pentadiagonal, the solution to this set of equations can be computed efficiently in $O(n)$ time using LU decomposition and backsubstitution. Note that the LU decomposition need be recomputed only when α changes.

In practice α is computed with an initial step size Δ_p, expressed in pixels, and the following formula computes the viscosity:

$$\alpha = \frac{\sqrt{2n}}{\Delta_p}\left|\frac{\partial \mathcal{E}}{\partial S}\right|, \tag{8}$$

where n is the number of vertices. This ensures that the initial displacement of each vertex is on the average of magnitude Δ_p. Because of the non linear term, we must verify that the energy has decreased from one iteration to the next. If, instead, the energy has increased, the curve is reset to its previous position, the step size is decreased, and the viscosity recomputed accordingly. This is repeated until the step size becomes less than some threshold value. In most cases, because of the presence of the linear term that propagates constraints along the whole curve in one iteration, it takes only a small number of iterations to optimize the initial curve.

The snakes described above have proved very effective at modeling smooth curves. Some objects, however, such as buildings, are best modeled as polygons with sharp corners. They can be handled in this context by completely turning off the smoothness term. Such objects typically have a relatively small number of corners, and the optimization is performed using a standard optimization technique.

3.2.2 Magnetic Snakes

Snakes usually converge to the intended image feature whenever the initial seed is "close enough" to the intended result. However, situations inevitably arise in which the snake will "get stuck" on some nearby feature corresponding to a true but unwanted local minimum of the objective function, thus preventing the extraction of the intended feature. In our current implementation, the user has no recourse except to abort the optimization and try again, either with a new seed curve, or with a new set of snake parameters.

A more natural mechanism for the user to interact with snakes is to allow him to use a mouse-controlled cursor to nudge the snake off the unintended feature. We have extended our formulation of the snake equations to include a term related to the length and direction of the

vector from each snake vertex to the mouse-cursor. In this way, we have simulated a magnetic force between the cursor and the snake which allows the user to push the curve in a desired direction.

3.3 Topology

An ordinary snake, represented as a polygonal curve defined by a sequence of 2-D vertices, is useful for modeling such features as vegetation boundaries and coastlines. A wider range of objects can be modeled by using other topological representations. For example, closed polygonal curves can model regions such as lakes and parking lots. In this section we describe two additional topologies that we have introduced: *3-D curves*, which can be instantiated using multiple views of a feature; and *ribbon-snakes*, in which each vertex has a width in addition to its position, which is useful for modeling roads and rivers.

3.3.1 3–D Linear Snakes

Snakes can be naturally extended to three dimensions by redefining $\mathcal{C}$ as a 3–D curve with n equidistant vertices $S = \{(x_i\ y_i\ z_i)\},\ i = 1, \ldots, n\}$ and considering its projections in a number of images for which we have accurate camera models. The average edge strength $\mathcal{G}(\mathcal{C})$ of Equation 3 becomes the sum of the average edge strengths along the projection of the curve in the images under consideration, and the regularization term of Equation 5 becomes

$$\begin{aligned} \mathcal{E}_D(\mathcal{C}) &= \mu_1 \sum_i (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 + (z_i - z_{i-1})^2 \\ &+ \mu_2 \sum_i (2x_i - x_{i-1} - x_{i+1})^2 + (2y_i - y_{i-1} - y_{i+1})^2 + (2z_i - z_{i-1} - z_{i+1})^2 \end{aligned} \tag{9}$$

Since the derivatives of $\mathcal{E}_D$ with respect to x, y, and z are still decoupled, we can rewrite Equation 7 as a set of three differential equations in the three spatial coordinates:

$$\begin{aligned} (K + \alpha I)X_t &= \alpha X_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial X}\right|_{X_{t-1}} \\ (K + \alpha I)Y_t &= \alpha Y_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial Y}\right|_{Y_{t-1}} \\ (K + \alpha I)Z_t &= \alpha Z_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial Z}\right|_{Z_{t-1}} \end{aligned}$$

where X,Y, and Z are the vectors of the x,y, and z vertex coordinates.

The only major difference with the 2–D case is the use of the images' camera models. In practice, $\mathcal{G}(\mathcal{C})$ is computed by summing gradient values along the line segments linking the vertices' projections. These projections, and their derivatives, are computed from the state vector S using the camera models. Similarly, to compute the viscosity, we use the camera models to translate the average initial step Δ_p, a number of pixels, into a step Δ_w expressed in world units and use the latter in Equation 8.

3.3.2 Ribbon Snakes

2–D snakes can also be extended to describe ribbon-like objects, such as roads, in aerial images. A ribbon snake is implemented as a polygonal curve, forming the center of the road. Associated with each vertex i of this curve is a width w_i that defines the two curves that are the candidate road boundaries. The state vector S becomes the vector $S = \{(x_i\ y_i\ w_i)\},\ i = 1, \ldots, n\}$ and the average edge strength is the sum of the edge strengths along the two boundary curves. Since the width of roads tend to vary gradually, we add an additional energy term in the form

$$\begin{aligned} \mathcal{E}_W(\mathcal{C}) &= \sum_i (w_i - w_{i-1})^2 \qquad (10) \\ \Rightarrow \frac{\partial \mathcal{E}_W}{\partial W} &= LW, \end{aligned}$$

where W is the vector of the vertices' widths and L a tridiagonal matrix. The total energy can then be written as

$$\mathcal{E}_(\mathcal{C}) = \lambda_D \mathcal{E}_D(\mathcal{C}) + \lambda_W \mathcal{E}_W(\mathcal{C}) - \lambda_G \mathcal{G}(\mathcal{C})$$

and at each iteration the system must solve the three differential equations:

$$\begin{aligned} (K + \alpha I)X_t &= \alpha X_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial X}\right|_{X_{t-1}} \\ (K + \alpha I)Y_t &= \alpha Y_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial Y}\right|_{Y_{t-1}} \\ (K + \alpha I)W_t &= \alpha W_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial W}\right|_{W_{t-1}} \end{aligned}$$

2–D ribbons can be turned into 3–D ones in exactly the same way 2–D snakes are turned into 3–D ones. The state vector S becomes the vector $S = \{(x_i\ y_i\ z_i\ w_i)\},\ i = 1, \ldots, n\}$ and at each iteration the system must solve four differential equations, one for each coordinate.

3.4 Optimization

A traditional snake can be described as a curve $\mathcal{C}$ that can deform itself to minimize an energy term that is the sum of a photometric energy term that attracts it towards edges, and a regularization term that enforces geometric constraints. In our implementation we take the photometric energy $\mathcal{E}_P(\mathcal{C})$ to be

$$\mathcal{E}_P(\mathcal{C}) = -\frac{1}{|\mathcal{C}|} \int_0^{|\mathcal{C}|} |\nabla \mathcal{I}(\mathbf{f}(s))|\ ds$$

where $\mathcal{I}$ is the image gray-level, s the curvilinear abscissa along the curve $\mathcal{C}$ and $|\mathcal{C}|$ its length. In practice $\mathcal{E}_P(\mathcal{C})$ is discretized and can be written as a function of the x and y coordinates of the snakes vertices.

3.4.1 Optimizing smooth snakes

When dealing with smooth curves, the deformation energy can be taken to be quadratic so that the optimization proceeds by solving at every time-step a linear system of equations of the form

$$\begin{array}{lllllllll} (K+(\gamma+\mu)I) & X_t & = & (\gamma+2\mu) & X_{t-1} & - & \mu X_{t-2} & + & F_X \\ (K+(\gamma+\mu)I) & Y_t & = & (\gamma+2\mu) & Y_{t-1} & - & \mu Y_{t-2} & + & F_Y \end{array}$$

where K is a stiffness matrix, X_t and Y_t are the vectors of the x and y coordinates of the snakes vertices at time t, F_X and F_Y are the derivatives of $\mathcal{E}_P$ with respect to X and Y, and γ and μ are mass and viscosity coefficients. Solving this system of equations using LU decomposition tends to propagate constraints along the whole curve in one iteration. As a result, convergence is typically achieved within a few iterations provided the starting point is relatively close to the desired answer.

For comparison's sake we have attempted to optimize the snake's objective function using conventional conjugate gradient. The propagation of constraints along the snakes is much slower and, as a result, the convergence properties are almost systematically much worse.

3.4.2 Optimizing polygonal snakes

Polygonal snakes typically have fewer vertices than smooth snakes, but obey no specific internal geometric constraints. For a polygonal curve C, the only energy being optimized is the photometric energy $\mathcal{E}_P(C)$. There is consequently no stiffness matrix, and, in this case, conjugate gradient has proved somewhat superior to regular gradient descent, however not by much. The energy landscape is not quadratic, therefore, most of the theoretical benefit of using conjugate gradient appears to be lost.

Both methods can fail due to the presence of bad local minima. To alleviate this problem, an attempt was made to use simulated annealing by allowing the position of the vertices to move randomly within a given distance of their current position. While this sometimes lets the snake escape from undesirable local minima, it is typically both slow and unreliable. Providing the user with easy-to-use tools to nudge the snake out of those minima is a much more effective approach.

3.4.3 Optimizing rectilinear snakes

Rectilinear snakes are polygonal snakes whose edges are constrained to form ninety degree angles. In effect for each set of three consecutive vertices, we want to impose the constraint

$$c_i = (x_i - x_{i-1})(x_{i+1} - x_i) + (y_i - y_{i-1})(y_{i+1} - y_i) = 0$$

The simplest way to enforce this constraint is to define a deformation energy of the form

$$\mathcal{E}_D(C) = \sum_i w_i c_i^2$$

where the w_i are weights. In practice we take them to be

$$w_i = \frac{1}{((x_i - x_{i-1})^2 + (y_i - y_{i-1})^2) + ((x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2)}$$

so that the polygon's sides tend neither to shrink nor to dilate.

We can then optimize a total energy that is a weighted sum of $\mathcal{E}_D$ and $\mathcal{E}_P$. While this converges for initial positions that are very close to the desired answer, it tends to fail as soon as the starting point is a bit farther. This stems from a well known problem: to impose the constraint, the contribution of $\mathcal{E}_D$ must be large and as a result the optimizer tends to minimize $\mathcal{E}_D$, (the geometric constraints) while ignoring $\mathcal{E}_P$, (the image information).

To remedy this problem, a simple constrained optimization method was developed that seems well suited to the problem. Given a function f of n variables $X = \{x_1, x_2, .., x_n\}$ to be minimized under a set of m constraints $C(X) = \{c_1, c_2, .., c_m\}$, we form the nxm Jacobian matrix A,

$$A = \begin{vmatrix} \frac{\partial c_1}{\partial x_1} & \cdots & \frac{\partial c_m}{\partial x_1} \\ \cdots & \cdots & \cdots \\ \frac{\partial c_1}{\partial x_n} & \cdots & \frac{\partial c_m}{\partial x_n} \end{vmatrix} ,$$

and perform at every iteration the following two steps.

1. Take a Newton step to project the state variables onto the constraint surface. This is achieved by solving the linear system

$$A^t A dX = -C(X)$$

and incrementing X by AdX. $X + AdX$ should be close to the constraint surface because

$$0 = c(X + AdX) \approx c(X) + A^t A dx \ .$$

2. Minimize f in a direction parallel to the projection of its gradient onto the tangent plane to the constraint surface. To compute this direction, we first solve the linear system

$$A^t A \lambda = A^t \nabla f$$

and take the direction to be $\nabla f - A\lambda$. It satisfies our requirement because λ is the least-squares solution of $A\lambda = \nabla f$ and $A\lambda$ is therefore the component of ∇f that is normal to the constraint surface.

We have used our rectilinear snakes to test this method and have found a notable improvement in convergence properties over the gradient and conjugate gradient methods.

3.4.4 Optimizing 3–D snakes

3–D snakes are those whose vertices are defined by their position in 3–D space and whose photometric energy is computed by summing the photometric energies of their 2–D projections in two or more images.

The observations made for the 2–D snakes carry over to 3–D. For smooth curves, the traditional snake approach to optimization is clearly the best while for 3–D rectilinear snakes, the constrained optimization method clearly wins. This method has also been used successfully to enforce a planarity constraint on 3–D polygonal snakes, thereby improving their convergence properties by reducing the numbers of degrees of freedom.

3.5 Initial Conditions

All model-based optimization primitives require an initial configuration to be used as the starting point for an iterative optimization procedure. The initial configuration is important because it determines which, of the many local minima that may exist in the landscape defined by the objective function, will be found. Normally, the initial configuration is provided by the user in an interactive setting. The initial configuration may be coarse, imprecise, and tedious to provide. Automated means for producing the initial configurations are desired, as are procedures that allow the use of less burdensome initializations.

Successful optimization of a convoluted or highly curved image feature requires specification of a large number of seed points along the length of the curve. While this polygonal approximation need not be specified precisely, there must be a sufficient number of vertices to prevent the snake from converging to an undesired local minimum. This behavior clearly reduces the potential benefit from the employment of snakes for modeling roads, rivers, and other curvilinear features.

Recent work by Walter Neuenschwander at Eidgenossische Technische Hochschule (ETH), Zurich Switzerland and Pascal Fua at SRI, has led to the development of a new optimization procedure that reduces the required number of seed points. A preprint of a technical paper describing the approach is included as appendix A. The procedure requires only the designation of the endpoints of the curve (and their tangents), and attempts to find the best image curve connecting the endpoints. The optimization proceeds in sequential fashion from the endpoints toward the center, taking image constraints into account first, only at the extremities, and then, progressively toward the center. If the evaluation of this approach (which has yet to be conducted) is promising, this technique should significantly reduce the length of time required to extract 3D models of roads, rivers, coastlines and railroad tracks for inclusion in RADIUS site models.

3.6 Snake Genes

The parameters associated with implementation of the snake algorithm are shown in Table 2. The approach to defining snake species has been to identify a number of terms to be included in the snake's objective function (Equation 1). Each term imparts a particular bias upon the snake, typically toward a particular geometric configuration or with an affinity for certain image structures. The terms can be mixed and matched, thereby allowing the definition of a large number of individual species through a combination. These individual objective function terms are viewed as *genes*; their presence or absence determining the makeup of the individual snake. Table 1 lists the objective function terms (the genes) and the most useful of their possible values.

3.7 Snake Species

The snake genes listed in Table 1 are the building blocks for assembling snakes that are tailored to the extraction of particular feature classes. By considering all possible combinations of snake genes, we have identified the features that can be extracted.[1] The results of that process are

[1] Even though the appropriate constraints can be modeled mathematically, effective mechanisms for optimizing the objective functions may not exist. Continuing research is devoted to finding optimization procedures that

Table 2: Snake parameters

Snake parameters	Description
Type of snake	Snakes can model smooth, polygonal or ribbon curves
Smoothing term	Whether or not to enforce a smoothing constraint
Optimization procedure	Gradient descent, conjugate gradient, simulated annealing, ...
Fixed endpoints	The endpoints of the snake can be either fixed or not
Gaussian smoothing	Size of the gaussian mask used to compute image gradients
Initial step size	Δ_p, pixel step size of Equation 8 used to compute the initial viscosity
Stick length	Initial intervertex spacing of the snake, in pixels
Smoothness constraint	λ'_D weight of the deformation component, Equation 5
Width constraint	λ'_W weight of the width component, Equation 10
Curvature/tension ratio	Relative contribution of tension and curvature, Equations 5 and 9

summarized in Table 3.

The species of snake is denoted by a sequence of these four characters:

1. Curvature: S = Smooth-curve
   P = Polygonal
   R = Rectilinear
2. Closure: O = Open
   C = Closed
3. Width: T = Thin
   F = Fixed-width
   V = Variable-width
4. Photometry: E = Edge
   L = Line

For example, SOTE denotes a smoothly curved, open-ended, thin snake with an affinity for aligning itself with edges in the image. All of these species can be instantiated as either 2–D snakes (for use with monocular imagery), or as 3–D snakes (for use with stereo pairs or other multiple images simultaneously).

Table 4, identical to Table 3, is reorganized by feature class. Table 4 shows the snake species that are applicable for each feature of interest to an image analyst. It illustrates the wide range of features that can be extracted by using snakes, however there are additional features for which no currently defined snake species is suitable. Additional species for some of these classes (e.g., fuel storage tanks) could be defined, however the effort is not justified.

work reliably for the snake species listed in the table.

Table 3: Snake species as defined by their genes.

Species	Feature classes
SOTE	Vegetation boundary, coastline, ridgeline
SOTL	Drainage, road in low-res imagery, wall
SOFE	Railway bed
SOFL	Railroad track, highway lane
SOVE	Road, highway, stream, river
SOVL	Road, highway
SCTE	Forest, field, lake, pond
SCTL	Lake, pond
SCFE	Track
SCFL	–
SCVE	–
SCVL	–
POTE	Fence, retaining wall, pier
POTL	Fence, powerline, telephone wire, pipeline
POFE	Runway, canal
POFL	Powerlines
POVE	Street, sidewalk
POVL	Street, sidewalk
PCTE	Parking lot, storage area, pasture
PCTL	–
PCFE	–
PCFL	–
PCVE	–
PCVL	–
ROTE	–
ROTL	–
ROFE	–
ROFL	–
ROVE	–
ROVL	–
RCTE	Roof, parking lot, pier
RCTL	Roof, parking lot
RCFE	–
RCFL	–
RCVE	–
RCVL	–

Table 4: Snake species sorted by feature class.

Feature	Snake species
Canal	POFE
Coastline	SOTE
Drainage	SOTL
Fence	POTE, POTL
Field	SCTE
Forest	SCTE
Highway	SOFL, SOVE, SOVL
Lake	SCTE, SCTL
Parking lot	PCTE, RCTE, RCTL
Pasture	PCTE
Pier	POTE, RCTE
Pipeline	POTL
Pond	SCTE, SCTL
Powerline	POTL, POFL
Railroad track	SOFL
Railway	SOFE
Ridgeline	SOTE
River	SOVE
Road	SOTL, SOVE, SOVL
Roof	RCTE, RCTL
Runway	POFE
Sidewalk	POVE, POVL
Storage area	PCTE
Stream	SOVE
Street	POVE, POVL
Telephone wire	POTL
Track	SCFE
Vegetation boundary	SOTE
Wall	SOTL, POTE

3.8 Snake Examples

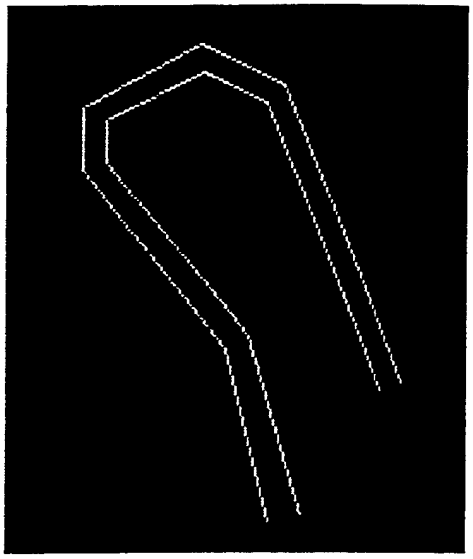
Figures 1 and 2 show sub-images of two sites we have used in our tests. A site consists of several images, generally aerial images of dimensions greater than 1000×1000 pixels. The two test sites differ from on another. The first is a mountainous rural area with several industrial facilities (Figure 1.a), while the second is an urban area on flat terrain (Figure 2.a).

Examples of the use of two of our snake species are illustrated by the following figures. Figures 1.b and 2.b show curves used as seeds in the snake optimization process. Figures 1.c and 2.c show the results of the optimization.

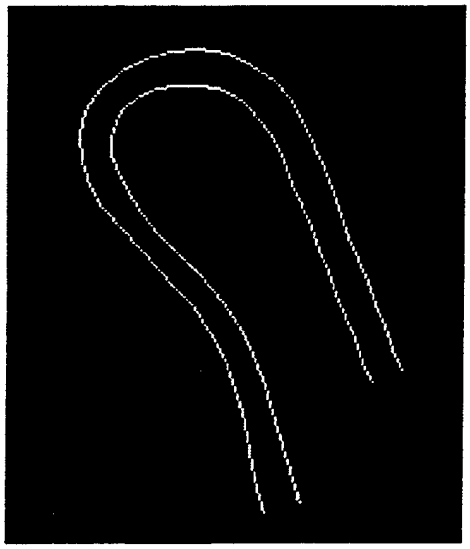
The task illustrated by Figure 1 is to extract the boundaries of the dirt road. From Table 4 we find that the appropriate snake species are SOTL, SOVE, and SOVL. SOVE was used because it is a smooth, open-ended curve with variable width demarked by step edges in the intensity image. SOTL was not used because it is only appropriate for roads appearing in low-resolution imagery. SOVL was not used because the sides of the road are characterized by a change in intensity rather than stripes along the boundaries. Eventually the species and its parameters will be chosen on the basis of context. It is currently done manually. The parameters actually used to produce the road model portrayed in Figure 1.c are listed in Figure 1.d.

In Figure 2, the task is to extract a precise boundary of the roof. The applicable species (from Table 4) are RCTE and RCTL. RCTE was used because the appearance of the roof perimeter in the image is characterized by step edges better than by thin lines. Although the building boundaries presented in Figures 2.b and 2.c appear similar, careful inspection reveals that there are significant differences between the two — the optimized version is much more precise than the sketch.

(a)



(b)

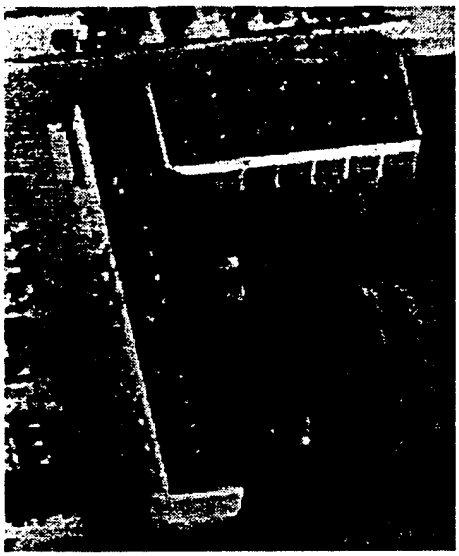


(c)

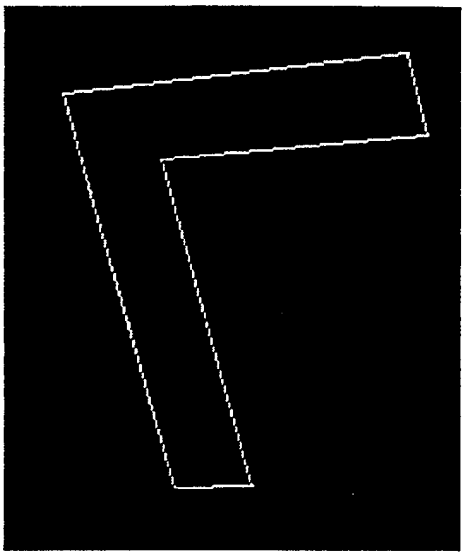
Parameters	Values
Type of snake	SOVE
Fixed endpoints	true
Gaussian smoothing	2
Initial step size	2.0
Stick length	10
Smoothness constraint	0.6
Width constraint	0.5
Curvature/tension ratio	1.0

(d)

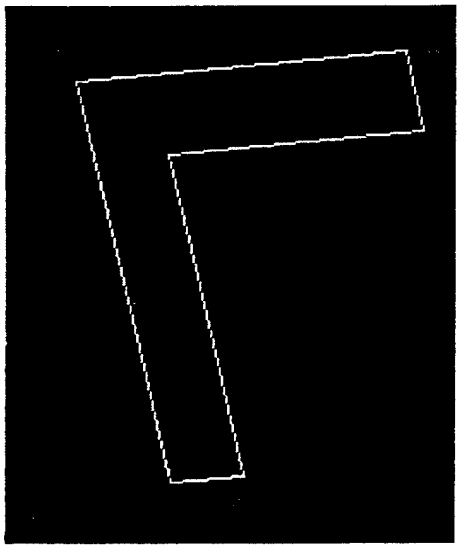
Figure 1:
(a) Example of images of the first test site. (b) Ribbon seed curve. (c) Snake-opimized ribbon curve. (d) Parameters used.



(a)



(b)



(c)

Parameters	Values
Type of snake	RCTE
Fixed endpoints	not used
Gaussian smoothing	1
Initial step size	2.0
Stick length	not used
Smoothness constraint	not used
Width constraint	not used
Curvature/tension ratio	not used

(d)

Figure 2:
(a) Example of images of the second test site. (b) Closed 2–D curve. (c) Snake-opimized 2–D curve. (d) Parameters used.

4 Context-Based Vision

Developers throughout the RADIUS Program are devising IU algorithms for use in site-model construction and change detection [13, 14, 15, 16]. SRI, is investigating the design of an architecture that can be used as the basis to control the invocation of IU algorithms for feature extraction. The key research question is whether sufficient contextual constraints are available to choose the algorithms and parameters that are necessary for site model construction.

The approach has been to apply the context-based architecture incorporated in (CONDOR)[2], a system that automatically constructs scene models of natural terrain from ground-level views. The semiautomated nature of RADIUS allows access to additional sources of contextual constraints that were not available to CONDOR. The CONDOR mechanisms are being adapted to better suit the interactive nature of RADIUS. A technical paper on the current state of that research was published in the Proceedings of the 1993 ARPA IU Workshop [17].

This portion of work was originally scheduled to be the focus for Year 3 of the project, however, its schedule has been accelerated to allow early incorporation into the RADIUS Phase 2 Testbed.

4.1 Motivation

Thirty years of research in image understanding have produced an enormous number of computer vision algorithms, many of which have demonstrated reliable performance at solving particular tasks in restricted domains, however, the development of computer vision systems that are reliable in more general circumstances has proved elusive. It is in response to this situation that a framework is proposed within which computer vision algorithms of specialized competence can be used and integrated with other such algorithms to produce a reliable vision system that operates effectively in a broader context than any of its individual component algorithms can.

The image understanding system envisioned comprises a large number of computer vision algorithms, each tailored to accomplish a particular task under a particular set of circumstances. The goals of these algorithms may overlap or even duplicate each other's goals, but the assumptions that they make and the data with which they operate will often differ.

The strategy for integrating such a collection of computer vision algorithms is to represent explicitly the assumptions made by each algorithm, and to use the context of the present task to select the most appropriate algorithms for solving that task. By doing so we hope to avoid the source of many failures of computer vision techniques — the employment of an algorithm outside the bounds of its intended domain of competence.

4.2 Background

In the emerging context-based vision paradigm, the development of general-purpose visual capabilities is attempted by assembling large numbers of special-purpose algorithms. Their invocation and interpretation of results are mediated by the consideration of contextual information.

[2]for Context-Driven Object Recognition

The CONDOR image understanding system was designed along these lines to serve as the perceptual architecture for a hypothetical outdoor robot. Given an image and a possibly extensive database describing the robot's environment, the system is to analyze the image and to augment its world model. CONDOR's recognition vocabulary consists mainly of natural objects such as trees, bushes, trails, and rocks. Because of the difficulty of recognizing such objects individually, CONDOR accepts an interpretation only if it is consistent with its world model. CONDOR recognizes entire contexts, rather than individual objects [18, 19, 20].

By making explicit the built-in assumptions inherent in all computer vision algorithms, the CONDOR architecture has been designed to allow context to influence the recognition process. In the context-based vision paradigm, the invocation of all algorithms is governed by context. Rather than employing a hard-wired control structure, the process is driven by context.

CONDOR associates a data structure called a *context set* with each IU algorithm. The context set identifies those conditions that must be true for that algorithm to be applicable. Efficient and effective visual recognition can be achieved only by invoking IU algorithms in those contexts in which they are likely to succeed.

Formally, a context set is a collection of context elements that are sufficient for inferring some relation or applying some algorithm. A *context element* is a predicate involving any number of terms that refer to the physical, photogrammetric, or computational context of image analysis.

Each algorithm has an associated context set, and is invoked only if its context set is satisfied. A context set is considered to be satisfied only if all its context elements are satisfied. As an example, consider a simple operator that extracts blue regions to find areas that could be labeled "sky." A context set for this operator might be

{ image-is-color, camera-is-horizontal, sky-is-clear, time-is-daytime }

The blue-sky algorithm would be unreliable if it were employed in anything but this context.

4.3 Implementation Choices

While CONDOR has demonstrated a significant capability for recognizing natural objects in ground-level outdoor imagery, perhaps its more enduring contribution lies in its context-based architecture (CBA), which offers a design methodology with broad applicability. For example, CBA is an attractive framework for organizing a site-model construction system using many algorithms that extract different features under different circumstances. This report describes the use of CBA in a site-model construction system.

Several alternatives exist for implementing the context-set concept.

4.3.1 Context Sets

Context sets are used to specify the conditions that must be met for a given algorithm to be applicable. The context set can also specify the conditions that must be met for a given parameter setting to be useful. For example,

MBO(closed-curve, rectangular-corners, manual-entry, gradient-descent)

specifies the parameters for a model-based optimization (MBO) algorithm that could be used to extract roof boundaries under some circumstances. The following context set encodes conditions

that are required for the extraction of roofs using that algorithm:

{ image-is-bw, image-resolution$\leq$ 3.0, interactivity-is-semiautomated }

This context set gives the requirements that must exist for the MBO algorithm to be applicable and it specifies the suitable parameter values. In the previous example for detecting roofs, the parameters were specified as having a closed-curve topology, an objective function preferring rectangular corners, initial boundary provided by manual entry, and the use of a gradient-descent optimization procedure.

In practice, a large number of context sets governing the application of MBO algorithms, as well as other algorithms, could be constructed and used to implement a cartographic feature-extraction system suitable for site-model construction. It is clear that such a collection could be unwieldy and difficult to maintain. A more structured representation of the context-set concept is needed.

4.3.2 Context Tables

One alternative representation for context sets is the *context table* — a data structure that tabulates the context elements in a more structured fashion [17]. An IU algorithm is associated with each row in the table; each column represents one context element.

The context table is equivalent to a collection of context sets. Conceptually, it provides a more coherent view of the contextual requirements of related algorithms. Applicable algorithms are selected by finding rows for which all conditions are met. One drawback to the table representation is its potentially large size. Each algorithm may require many rows to capture the contextual constraints of its various parameter combinations. Its chief value is its organization of contextual information for knowledge-based construction.

4.3.3 Context Rules

A third alternative for representing context sets is to encode them as production rules whose antecedent is the context set, and whose consequent is the applicable algorithm. For example,

{ image-is-bw, image-resolution$\leq$ 3.0, interactivity-is-semiautomated } $\Longrightarrow$
MBO(closed-curve, rectangular-corners, manual-entry, gradient-descent):

One advantage of encoding the rules as a logic program is that using the logic program interpreter eliminates the need to devise special machinery to test satisfaction of context sets. Unification provides a generic mechanism for matching the constraints embodied in a rule to the current context. Context rules can be more compact than the equivalent context table because additional predicates can be introduced to capture common context elements that appear in multiple rows of the table.

Whatever representation is chosen, it is clear that context sets can be employed in either direction. In the forward direction, the context sets are used to find applicable algorithms. In the opposite direction, the sets can be used for several purposes, including the selection of images on which to invoke a given algorithm.

4.4 Context-Based Architecture Specification

We have chosen to use logic programming to implement our context-based architecture, for the reasons discussed in the previous section. In this document, we adopt the Prolog syntax to illustrate the concepts [21]. It should be noted that the choice of logic programming (and Prolog in particular) is primarily a matter of implementation convenience — other schemes could be employed as effectively.

4.4.1 Terms

Before describing the context-based architecture in detail, we introduce two concepts that are central to its operation: algorithms and tasks (Figure 3).

```
algorithm(Name, Parms)
  Name:  The name of a lisp function
  Parms: A list of the parameters to the function

task(Name, Feature, Given)
  Name:    A generic task name, such as extract or refine
  Feature: A feature type  (eg, :building, :road, :functional-area)
  Given:   An object to be used, an image, a region, or other designation of
           a location where the task is to be accomplished
```

Figure 3: Documentation of compound terms.

An *algorithm*, represented by a logical term, designates a piece of code that implements a computer vision technique. Each algorithm term indicates the name of the procedure to be invoked and a list of the parameters it expects. The parameters will typically include a list of the images to be employed, any initial conditions to be provided, and the typical arguments to the function. For example,

```
algorithm(road-tracker, (ft-hood-2, 3d-ribbon-curve, (942, 1516)))
```

indicates that the road-tracker algorithm is to be invoked on the image named `ft-hood-2`, creating an object of class `3d-ribbon-curve` starting at image coordinate (942, 1516).

A *task* is a term that indicates an operation that could be performed. It includes the name of the task, the type of feature that is desired, and a list of the data to be used in accomplishing the task. An example task is

```
task(extract, wide-road, world-point(-5407.3, 5732.2)).
```

Here, the task is to extract a wide road starting from the specified image coordinate.

4.4.2 Predicates

Figure 4 documents the two most important predicates used in the context-based architecture: `solves` and `invoke` [3].

The predicate `solves(Alg, Task)` is true if the indicated algorithm can be used to solve the indicated task. The specification of the algorithm includes the specification of the imagery, parameters, and other given information that is to be used. The task provides a complete specification of the goal to be accomplished and the constraints to be exploited.

The predicate `invoke(Alg)` is true if the indicated algorithm is applicable to solving the current task. It is the truth of this predicate, and the associated binding of Alg, that the context-based architecure is expected to compute.

```
solves(Alg, Task)
  Alg is an algorithm that can be used to solve the task.
  Task is a specification of that task.

invoke(Alg)
  Alg is an algorithm that is applicable to solve the current task
      in the current context.
```

Figure 4: Documentation of some predicates.

4.4.3 Rules

The primary rule governing the operation of the CBA is shown in Figure 5(a). It specifies the conditions that must be satisfied in order for an appropriate algorithm to be found to solve the currently desired task in the current context. In words, this rule states that an algorithm to be invoked must be capable of solving the desired task, and that its level of interactivity must be what is desired and its accuracy must meet the desired accuracy. The terms involving the predicates `desired-task(Task)`, `desired-interactivity(Int)`, and `desired-accuracy(Acc)` are expected to be provided in advance by the user.

Several auxiliary rules, such as the one shown in Figure 5(b), add versatility to the control structure. This rule states that an algorithm that solves a task using a particular image also solves that task for the site captured by the image. For example, the task

```
task(extract, buildings, fort-hood)
```

can be reduced to

```
task(extract, buildings, fort-hood-image1)
```

if `fort-hood-image1` actually portrays the site, Fort Hood.

[3] Throughout this paper, normal Prolog syntax is used. By convention, the names of all variables are capitalized, while predicates and ground atoms begin with a lowercase letter.

```
(a)
%  invoke(Alg)
%       Alg is an algorithm that could be invoked to solve
%       the given task in the specified context.

invoke(Alg)  :- solves(Alg, Task),
                desired-task(Task),
                desired-interactivity(Int),
                interactivity(Alg, Int),
                desired-accuracy(Acc),
                accuracy(Alg, Acc).

(b)
%    An algorithm that solves a task using an image, also solves the task
%    for the site of the image.
solves(Alg, task(Name, Feature, Site) )  :-
                solves(Alg, task(Name, Feature, Image)),
                image-site(Image, Site).
```

Figure 5: Documentation of the top-level rules.

Posing the query `invoke(Alg)` triggers the search of a binding for Alg that solves the desired task, `Task`. If successful, the binding of `Alg` that is produced is typically a compound term specifying an algorithm, the imagery to be used, and any parameters it needs. If no applicable algorithm can be found, the search terminates and the failure is reported.

The use of the Chain Rule given in Figure 6(a) allows the interpreter to search for a sequence of algorithms that collectively solve the desired task. The rules in Figure 6(b), (c), and (d) describe how the interactivity and accuracy of a composition of algorithms relate to the properties of each individual algorithm.

4.4.4 Rule Packets

Introduction of a new algorithm to the system requires specification of a packet of rules that delimit its scope of applicability, its properties, and its assumptions.

For example, Lynn Quam's Road Tracker algorithm [22] seeks to follow the extent of a road by correlating successive cross sections starting from an initial seed point. It requires that the width of the road occupy at least 4 pixels. The rule packet governing the application of this algorithm is given in Figure 7. The first rule states that the algorithm can be used to extract any feature that is a subclass of type road, that the image it uses must have a ground-sample-distance of less than 1.5 meters (i.e., a 6-meter-wide road must occupy at least 4 pixels), and that the algorithm will use an instance of an RCDE class that is suitable for modeling the type of feature desired.

The remaining two rules specify that the road-tracker algorithm is inherently semiautomatic (because it requires the user to provide an initial seed point), and that the delineation accuracy

```
(a)
% Chain rule:
%   A task can be solved by splitting it into two parts.
solves(algorithm(compose, [Alg1, Alg2] ), task(extract, Feature2, Image) )  :-
        solves(Alg1, task(extract, Feature1, Feature2)),
        solves(Alg2, task(extract, Feature2, Image)).

(b)
% The interactivity of the composition of two algorithms with the
% same interactivity is the same.
interactivity(algorithm(compose, [Alg1, Alg2] ), Int)  :-
        interactivity(Alg1, Int),
        interactivity(Alg2, Int).

(c)
% If either algorithm is manual or semiautomatic, the composition
% must be semiautomatic.
interactivity(algorithm(compose, [Alg1, Alg2] ), semiautomatic)  :-
        interactivity(Alg1, Int1),
        interactivity(Alg2, Int2),
        =\(Int1, Int2).

(d)
% The accuracy of the composition of two algorithms is the accuracy
% of the last algorithm
accuracy(algorithm(compose, [Alg1, Alg2]) , Acc)  :-
        accuracy(Alg1, Acc).
```

Figure 6: The Chain Rule.

```
%  road-tracker(Image, Class)
%     The top-level lisp function to invoke Lynn Quam's road tracker algorithm.
%     This version is interactive -- it prompts the user to provide
%                                    the initial pair of points.
%        Image:  The image to be used
%        Class:  The RCDE class to be instantiated


solves(algorithm(road-tracker, Image, Class), task(extract, Feature, Image))  :-
        image(Image),
        subtype(road, Feature),
        rcde-class-of-feature-type(Class, Feature),
        gsd(Image, X),
        <(X, 1.5).


interactivity(algorithm(road-tracker, Image, Class), semiautomatic).

accuracy(algorithm(road-tracker, Image, Class), coarse).
```

Figure 7: The rule packet for the Road Tracker algorithm.

that can be expected is qualitatively designated as "coarse."

4.5 Examples

We have implemented a prototype of the context-based architecture using Prolog. This software is presently being tested and extended to provide more flexibility and more powerful constructs for use in the algorithm-specific rule packets.

- Mouse input of points and lines
- Interactive RCDE object-modeling primitives
- A subset of the model-based optimization algorithms
- The Quam Road Tracker
- The Cookie-Cutter algorithm for automatically cloning structures

The rule packets for these algorithms illustrate the basic operation of the CBA. More advanced concepts are currently being devised and implemented to allow more precise specification of when an algorithm should be used and how its parameters are to be determined.

```
(a)
image-site(fort-hood-image1, fort-hood).
image-site(fort-hood-image2, fort-hood).
image-site(fort-hood-image3, fort-hood).

gsd(fort-hood-image1, 2.0).   % image resolution, in meters
gsd(fort-hood-image2, 2.0).
gsd(fort-hood-image3, 0.8).

subtype(road, one-lane-road).     % feature-type hierarchy
subtype(road, two-lane-road).

                                  %  association of modeling primitives
rcde-class-of-feature-type(cube-object, building).
rcde-class-of-feature-type(3d-ribbon-curve, one-lane-road).
rcde-class-of-feature-type(3d-ribbon-curve, two-lane-road).


(b)
desired-task(extract, two-lane-road, fort-hood).
desired-accuracy(coarse).
desired-interactivity(semiautomatic).
```

Figure 8: A set of facts used to answer an example query.

Each algorithm that is to be controlled by the CBA is represented by a packet of rules. Section 4.5 contains rule packets for the following algorithms:

To illustrate the operation of the CBA, consider the goal of semiautomatically extracting a model of a road, using the rule packets listed at the end of this section (beginning on Page 28). The complete logic program used by the CBA also contains a collection of facts about the current context. This includes a list of images that are available and their properties. In practice, these can be computed at run time through procedural attachment within the logic programming, but for illustrative purposes we list them here explicitly as a collection of facts (Figure 8(a)).

The user asserts additional facts as shown in Figure 8(b), to specify the task and the constraints on that task. The CBA is then activated by posing the query to the Prolog interpreter:

```
invoke(Alg)
```

to which the only acceptable binding for `Alg` is

```
algorithm(road-tracker, ft-hood-image3, 3d-ribbon-curve)
```

This example makes use of the rule in Figure 5(b) to find images of the Fort Hood site for further consideration. Only `fort-hood-image3` has sufficient resolution for use by the `road-tracker` algorithm. The example illustrates how the CBA has identified a suitable algorithm, its parameters, and an image, given only the specification of the task to extract a road from Fort Hood.

If more than one binding of `Alg` would have solved the query, the Prolog interpreter would compute each one of them. Additional machinery is currently being developed to allow the CBA to choose the best algorithm, or best imagery, when more than one solution exists.

Prolog Rule Packets for Several Algorithms

```
%MOUSE SPECIFICATIONS:

/*
mouse-get-point(Image, Prompt)
   A lisp function that prompts the user to mouse an image point
  Image:  The image to be displayed during input
  Prompt:  The test string to be displayed during input
*/

solves(algorithm(mouse-get-point, (Image, "Pick a point:") ), task(extract, point, Image) )  :-
        image(Image).

interactivity(algorithm(mouse-get-point, Parms), manual).

accuracy(algorithm(mouse-get-point, Parms), coarse).


%-------------------------------------------------------------------------

/*
mouse-get-point-pair(Image, Prompt)
   A lisp function that prompts the user to mouse an image point
  Image:  The image to be displayed during input
  Prompt:  The test string to be displayed during input
*/

solves(algorithm(mouse-get-point-pair, (Image, "Pick a pair of points:") ),
          task(extract, point-pair, Image) )  :-
        image(Image).

interactivity(algorithm(mouse-get-point-pair, Parms), manual).

accuracy(algorithm(mouse-get-point-pair, Parms), coarse).


%=========================================================================

% RCDE CREATE OBJECT:

/*
rcde(Image, Class)
   A lisp function that allows the user to create an RCDE ojbect of type Class
  Image:  The image to be displayed during input
  Class:  The RCDE class of object to be created
*/

solves(algorithm(rcde, (Image, Class) ), task(extract, Feature, Image) ) :-
        image(Image),
        feature-type(Feature),
        rcde-class(Class),
        rcde-class-of-feature-type(Class, Feature) .

interactivity(algorithm(rcde, Parms), manual).

accuracy(algorithm(rcde, Parms), coarse).


%=========================================================================
```

```
% SNAKE SPECIFICATION:

/*
snake(Image, Object, Parms)
   The top-level lisp function to invoke snakes
*/

solves(algorithm(snake, [Image, Object, default-parms] ), task(refine, Object, Image))  :-
        image(Image),
        rcde-object(Object).

interactivity(algorithm(snake, [Image, Object, Parms]), automatic)  :-  rcde-object(Object).

accuracy(algorithm(snake, [Image, Object, Parms]), fine)  :-  rcde-object(Object).


%-------------------------------------------------------------------------

solves(algorithm(snake, [Image, radius-canal, pofe] ), task(refine, canal, Image) ) :-
        image(Image).

solves(algorithm(snake, [Image, radius-shoreline, sote] ), task(refine, shoreline, Image) ) :-
        image(Image).

solves(algorithm(snake, [Image, radius-fence, pote] ), task(refine, fence, Image) ) :-
        image(Image).

solves(algorithm(snake, [Image, radius-fence, potl] ), task(refine, fence, Image) ) :-
        image(Image).


% ...

%-------------------------------------------------------------------------

solves(algorithm(snake, [Image, Class, default-parms] ), task(extract, Feature, Image))  :-
        image(Image),
        rcde-class-of-feature-type(Class, Feature).

interactivity(algorithm(snake, [Image, Class, default-parms]), semiautomatic)  :-  rcde-class(Class).

accuracy(algorithm(snake, [Image, Class, default-parms]), fine)  :-  rcde-class(Class).


%=========================================================================

% ROAD TRACKER SPECIFICATION:



%  road-tracker(Image, Class)
%     The top-level lisp function to invoke Lynn Quam's road tracker algorithm.
%     This version is interactive -- it prompts the user to provide
%                                    the initial pair of points.
%        Image:  The image to be used
%        Class:  The RCDE class to be instantiated


solves(algorithm(road-tracker, Image, Class), task(extract, Feature, Image))  :-
        image(Image),
        subtype(road, Feature),
        rcde-class-of-feature-type(Class, Feature),
        gsd(Image, X),
```

```
        <(X, 1.5).

interactivity(algorithm(road-tracker, Image, Class), semiautomatic).

accuracy(algorithm(road-tracker, Image, Class), coarse).

%=============================================================================

%COOKIE CUTTER:

/*
cookie-cutter(Image, Prototype, Region)
   The top-level lisp-function to invoke Aaron's implementation of the cookie cutter algorithm
  Image:  The image to be used
  Prototype:  An RCDE object to be used as a replica for the desired objects
  Region:  An image region within which to search for copies of the prototype
*/

solves(algorithm(cookie-cutter, (Image, Prototype, Region) ) , task(extract, Feature, Region) ) :-
        image(Image),
        feature-type-of-object(Feature, Prototype).


interactivity(algorithm(cookie-cutter, Parms), automatic).

accuracy(algorithm(cookie-cutter, Parms), coarse).
```

4.6 Discussion

The context-based architecture described was designed to enable the construction of large, reliable image understanding systems by integrating a collection of reliable, but specialized algorithms. Initial experiments indicate that progress has been made in this direction, but more formal testing of even larger assemblages of algorithms is necessary to fully validate this belief.

It is clear that the performance of an IU system that employs the context-based architecture could also be attained by integrating the same computer vision algorithms via more traditional methods. However, the explicit representation of contextual constraints affords a number of additional benefits that would be lost to a purely functional integration. These benefits include:

- The context-based architecture allows the user to specify the task to be accomplished, leaving the selection of specific algorithms to be decided by the system. For example, the user can state that he would like the system to construct a 3D model of a building, and the system would decide which of several building extraction algorithms would be most appropriate given the current imagery and auxiliary data that is available. The user can make effective use of the IU system while possessing little knowledge of the capabilities and limitations of the individual computer vision algorithms.

- The context rules are used to establish algorithm parameter settings, in the same way as they delimit the range of applicability. While computer vision algorithms can often compute their parameter settings from data at run time, the context rules provide a uniform means for all algorithms to specify how their parameters are to be determined.

- When building large systems in an evolutionary fashion, it can be difficult to add new capabilities without jeopardizing the integrity of existing capabilities. The modular decomposition of the context rules allows the developer to integrate a new algorithm by adding a packet of rules governing its use. No modification of existing code is required, and one need not worry about interaction between the new packet of rules and the existing rule set, because the new packet only concerns the invocation of the new algorithm.

- It is important to identify the imagery that is most likely to allow an algorithm to yield a desired result, rather than to choose an algorithm to run on a preselected image. The context rules already encode the information necessary to make this determination — they can be used to answer this question by fixing the algorithm and allowing the image to be a variable in the query. In fact, the context-rule base can be used to answer both questions simultaneously, finding the best combination of algorithms and images to satsify a given task.

5 Knowledge-Base Acquisition

Our proposed system is intended to select, parameterize, and apply image understanding algorithms from a library of existing routines within an MSE system. Our work with the CONDOR system showed that by understanding the context of the sensor data to be interpreted, we can use a collection of relatively weak algorithms to make reliable decisions. In our past work, the

translation table going from contexts to the selection of parameterized algorithms was compiled manually — an unacceptable approach for an operational application in which the system might have to be modified in the field by non-IU-experts. We recently have developed an automated (learning-based) methodology for performing this task. The Appendix to this report contains a paper [12] that describes the approach we have developed for context-based learning and presents experimental data illustrating its effectiveness. While this work must still be considered to be in the preliminary research stage, it offers promise as the basis for automated knowledge-based acquisition in an operational system.

6 Data Sets from TEC

In mid-April 1994, a tape containing additional high-resolution oblique images of portions of the site at Martin-Marietta, Colorado, was received.

6.1 Installation in RCDE

Additional facilities have been added to the RCDE to read the TEC format tapes and to install the imagery in the RCDE. In particular, some of the functions that have been defined are:

read-1993-tec-basic-image-params : This function reads the *name, element-size, x-dimension, y-dimension, minimum-value*, and *maximum-value* from a stream and returns them as multiple-values to lisp.

read-1993-tec-image-header : This function takes a pathname of a file containing a TEC header file and generates the corresponding RCDE objects associated with the camera model for the image. It returns a list of the following objects: *file-to-image-matrix, image-to-scanner-matrix, scanner-to-fiducial-matrix, lens-corrections, focal-length* (in mm), *camera-position, photo-to-ground-matrix, earth-mean-radius, reference-coordinate-system, local-to-long-lat-info, local-to-utm-info, image-to-camera-matrix*, and *camera-to-image-matrix.*

read-tec-1993-dem-header : This function takes a pathname of a file containing a TEC DEM header file and generates the corresponding RCDE objects associated with the coordinate transforms associated with the DEM. It returns a list of the basic image parameters associated with the DEM, plus the DEM-to-UTM coordinate transform matrix.

make-camera-model-from-tec-header : This function takes the header information as parsed by the function *read-1993-tec-image-header* and generates an RCDE perspective transform object to use as the camera model for the associated image.

build-tec-3d-world : This function constructs the local coordinate system to be used as a georeference for a TEC format data set. It accepts a name for the site, plus the latitude-longitude of the origin, and establishes the requisite transforms to freely shift between UTM, Lat-Long, and Cartesian coordinate systems.

build-tec-site-from-utm-referenced-data : This function uses *build-tec-3d-world* to construct the 3d-world for a TEC format data set. It also takes special care to link terrain

data that is specified on a UTM grid, so that no accuracy is lost and the terrain model need not be resampled.

These functions, and other supporting functions have been collected in the file *tec-header-input.lisp* and installed as an application on top of the RCDE. We will distribute it with future versions of the RCDE. RADIUS contractors who desire to use it before the next release are welcome to contact us directly.

6.2 Initial Site Model Construction

After installing the imagery in the RCDE, an attempt was made to model one of the building complexes as a benchmark in modeling future complex buildings.

The complex is shown in Figure 9. Using the collection of interactive manipulations provided by the RCDE, we were able to generate the model shown in Figures 10 and 11 in three minutes. An additional minute was spent to associate texture maps with some of the building faces and roofs. After 15 seconds of processing time on a Sparc-2, the synthetic image shown in Figure 12 was generated.

Experience with this data set shows that it is not too difficult to model small complexes of buildings such as this one using current RCDE capabilities. The model could be generated more quickly and accurately with the incorporation of additional constraints, such as aligning walls of adjacent buildings. A menu operation should be added to associate texture maps with all visible faces, so that this task is less tedious.

Full automation of the reconstruction of a building complex is beyond the current capabilities of our model-based optimization techniques (and any other building extraction system that we are aware of). This data set could serve as a useful "challenge" problem for those laboratories attempting to automate the extraction of building models from overhead imagery.

7 Summary

The RADIUS Program could benefit greatly from the use of more highly automated means for constructing and updating 3-D site models. Our research on model-based optimization has led to the development of a number of tools that increase the degree of automation and precision that is possible. These have been implemented in the RCDE and incorporated in the RADIUS Testbed. Further use of these tools within the Testbed will undoubtedly lead to new ideas and additional improvements in the site-model construction process.

Our work on the context-based vision paradigm has led to the development of a new architecture for integrating independently developed IU algorithms within a single system. The architecture provides the additional benefit to RADIUS of allowing the IA to specify a desired result, rather than a specific procedure to be followed, in carrying out an image exploitation task. This architecture is now being refined, and will be incorporated in the RADIUS Testbed before the end of the project.

References

[1] D. Gerson. Radius: The government viewpoint. In *ARPA Image Understanding Workshop*, January 1992.

[2] J. Mundy and P. Vrobel. The role of iu technology in radius phase ii. In *Unpublished*, May 1994.

[3] Thomas M. Strat and W. Doublas Climenson. Radius: Site model content. In *ARPA Workshop on Image Understanding*, November 1994.

[4] A. J. Hanson and L. Quam. Overview of the SRI Cartographic Modeling Environment. In *DARPA Workshop on Image Understanding*, pages 576–582, April 1988.

[5] J.L. Mundy, R. Welty, L. Quam, T. Strat, W. Bremmer, M. Horwedel, D. Hackett, and A. Hoogs. The RADIUS Common Development Environment. In *DARPA Workshop on Image Understanding*, pages 215–226, 1992.

[6] D. Terzopoulos, A. Witkin, and M. Kass. Symmetry-seeking models and 3D object reconstruction. *International Journal of Computer Vision*, 1:211–221, 1987.

[7] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

[8] P. Fua and A.J. Hanson. Objective functions for feature discrimination. In *IJCAI*, Detroit, August 1989.

[9] Y.G. Leclerc. Constructing simple stable descriptions for image partitioning. *International Journal of Computer Vision*, 3(1):73–102, 1989.

[10] P. Fua and A.J. Hanson. An optimization framework for feature extraction. *Machine Vision and Applications*, 4(2):59–87, Spring 1991.

[11] P. Fua and Y.G. Leclerc. Model driven edge detection. *Machine Vision and Applications*, 3:45–56, 1990.

[12] S. Houzelle, T.M. Strat, P.V. Fua, and M.A. Fischler. Using contextual information to set control parameters of a vision process. *in preparation*, 1994.

[13] R. Chellappa, L.S. Davis, D. DeMenthon, A. Rosenfeld and Q. Zheng. Site-Model-Based Change Detection and Image Registration. In *ARPA Image Understanding Workshop*, April 1993.

[14] L.S. Davis C.L. Lin X. Zhang C. Rodriguez A. Rosenfeld R. Chellappa, Q. Zheng and T. Moore. Site-Model-Based Monitoring of Aerial Images. In *DARPA Workshop on Image Understanding*, November 1994.

[15] J.A. Shufelt and D.M. McKeown. Fusion of monocular cues to detect man-made structures in aerial imagery. *CVGIP: Image Understanding*, 57(3):307–330, May 1993.

[16] A. Huertas C. Lin and R. Nevatia. Detection of buildings using perceptual grouping and shadows. In *Proc. IEEE CVPR-94*, Seattle, June 1994.

[17] Thomas M. Strat. Employing contextual information in computer vision. In *DARPA Workshop on Image Understanding*, 1993.

[18] T.M. Strat and M.A. Fischler. Natural object recognition: A theoretical framework and its implementation. *Proc. IJCAI-91*, August 1991.

[19] T. M. Strat and M. A. Fischler. Context-based vision: Recognizing objects using both 2d and 3d imagery. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 13(10):1050–1065, October 1991.

[20] Thomas M. Strat. *Natural Object Recognition*. Springer-Verlag, New York, 1992.

[21] Leon Sterling and Ehud Shapiro. *The Art of Prolog*. The MIT Press, Cambridge, Mass., 1986.

[22] Lynn H. Quam and Thoms M. Strat. Sri image understanding research in cartographic feature extraction. In *Proc. ISPRS2*, Munich, September 1991.

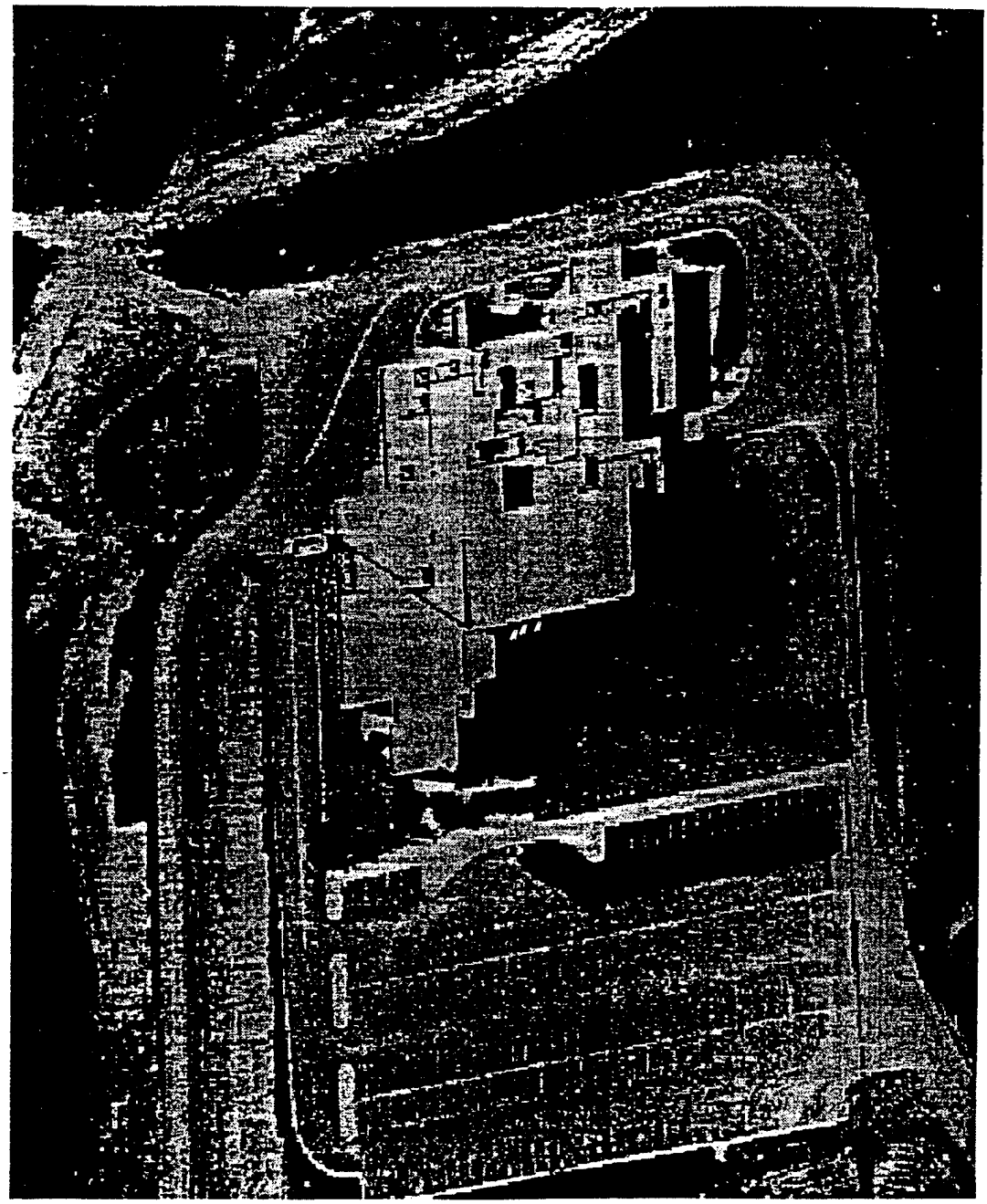


Figure 9: Image of building complex to be modeled.

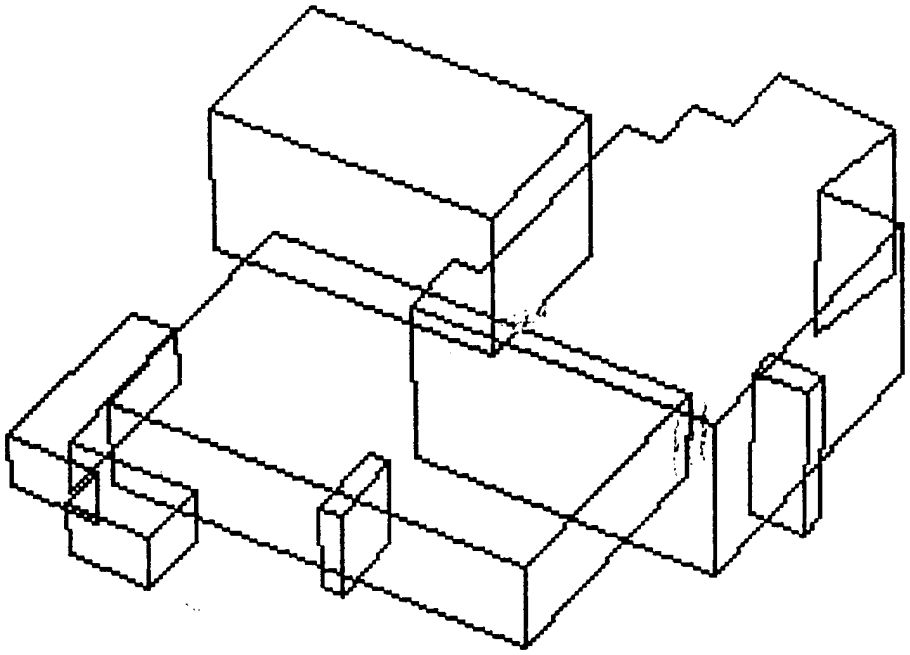


Figure 10: Wireframe model of building complex.

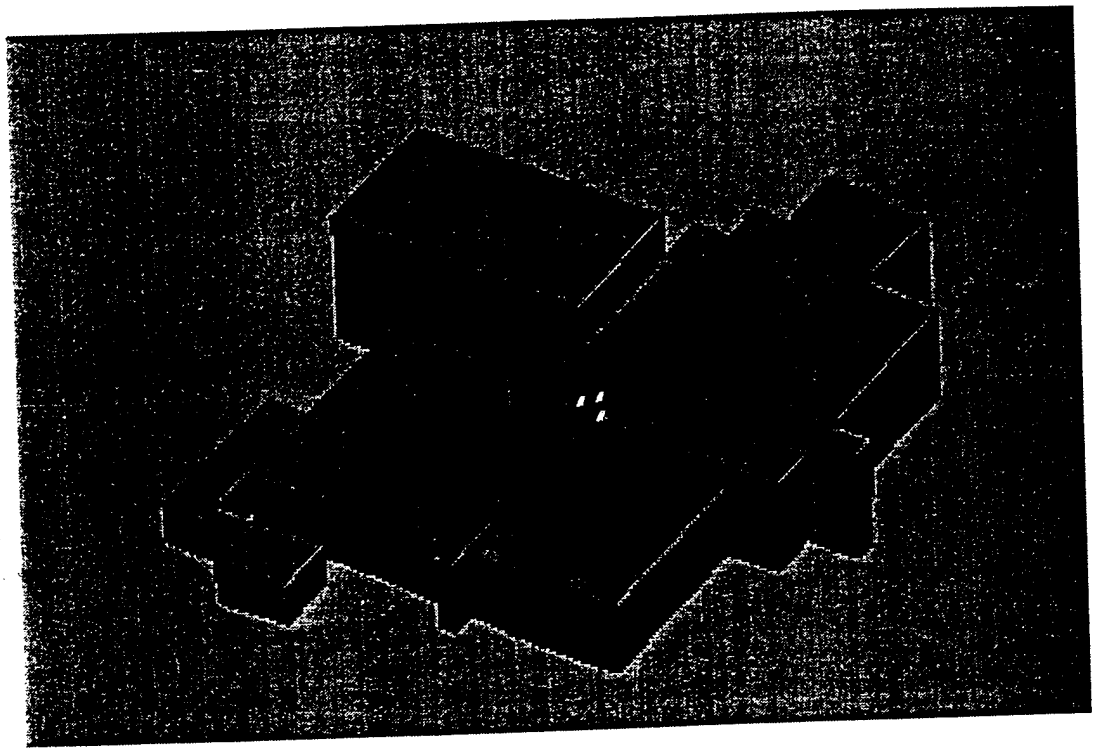


Figure 11: Shaded perspective view of building complex.

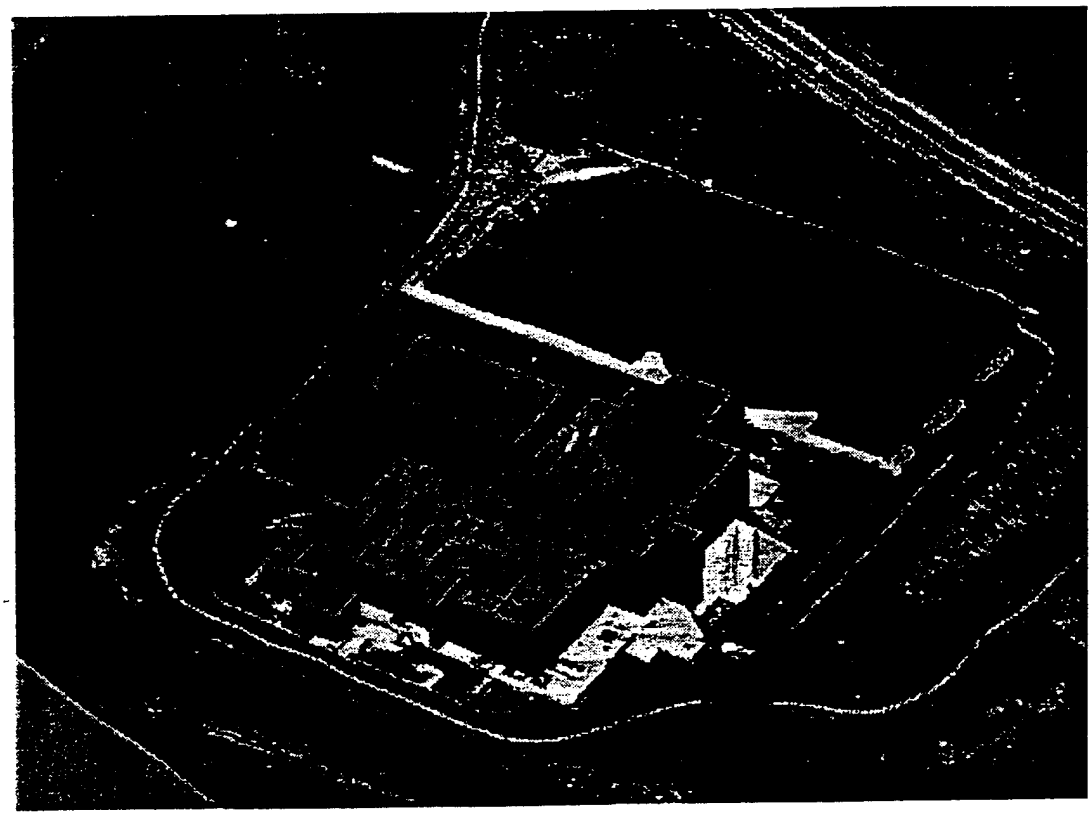


Figure 12: Synthetic perspective image of building complex.

Appendix A

"Initializing Snakes"

W. Neuenschwander, P. Fua, G. Szekely, and O. Kubler

published in

Proceedings, IEEE CVPR
Seattle, Washington
June 1994.

Initializing Snakes

W. Neuenschwander*, P. Fua†, G. Székely*, and O. Kübler*

* Communication Technology Laboratory
Swiss Federal Institute of Technology
ETH
CH-8092 Zurich, Switzerland

† SRI International
Artificial Intelligence Center
333 Ravenswood Avenue
Menlo Park, CA 94025, USA

Abstract

In this paper, we propose a snake-based approach that lets a user specify only the distant end points of the curve he wishes to delineate without having to supply an almost complete polygonal approximation. We achieve much better convergence properties than those of traditional snakes by using the image information around these end points to provide boundary conditions and by introducing an optimization schedule that allows the snake to take image information into account first only near its extremities and then, progressively, towards its center.

These snakes could be used to alleviate the often repetitive task practitioners have to face when segmenting images by abolishing the need to sketch a feature of interest in its entirety, that is, to perform a painstaking, almost complete, manual segmentation.

1 Introduction

In recent years snakes, have emerged as a very powerful tool for semiautomated object delineation. They have been originated by Terzopoulos, Kass, and Witkin [1, 2] and have since given rise to a large body of literature ([3, 4, 5, 6] among many others) that explores theoretical and implementation issues as well as new applications.

In most of these papers, however, it is assumed that the initial position of the snake is relatively close to the desired solution. While this is a reasonable assumption for applications such as motion tracking [7, 8], it is ineffective for delineating complex objects from scratch.

The optimization of the traditional snakes is typically global and takes edge-information into account along the whole curve simultaneously. When the snake's initial position is far away from the desired result, this often results in the snake getting stuck in an undesirable local minimum because it uses irrelevant edge information. The minimization problem is solved by treating the snake as a physical system evolving under the influence of the potential that is the sum of an objective function and a dissipation potential that enforces convergence. This potential tends to prevent the snake from moving far away from its current position, thereby contributing to forcing the snake's initial position to be close from the desired solution in order to achieve convergence and not get stuck in an undesirable local minimum. In the remainder of the paper, we will refer to these snakes as "dynamic snakes" because their position is computed by solving the dynamics equation of a physical system.

Here, we describe a snake approach that allows a user to specify only the end points of the curve he wishes to delineate instead of a complete polygonal approximation. This way we may abolish the need to outline the desired structure very precisely, that is, to perform a painstaking, almost complete, manual segmentation. The optimization progresses from the end points towards the center thereby effectively propagating edge-information along the curve without the need for a dissipation potential. The user-supplied end points and the automatically computed edge gradient in their vicinity serve as anchors. They are first used to compute an initial state that is approximately correct in the anchors' vicinity. While the image term is "turned on" progressively from the snake's extremities towards its middle section, the snake's position is iteratively recomputed. As a result, the snake eventually finds the smooth path, as defined by the regularization term, that best matches the edge connecting the two end points and has the right orientation at these points. We will refer to these snakes as "static" snakes.

In the following section, we describe the traditional or dynamic snakes. Next, we introduce our own breed of static snakes. Finally, we present results on both synthetic and real images that demonstrate the improved performance of the static snakes for initializations that are far away from the desired result.

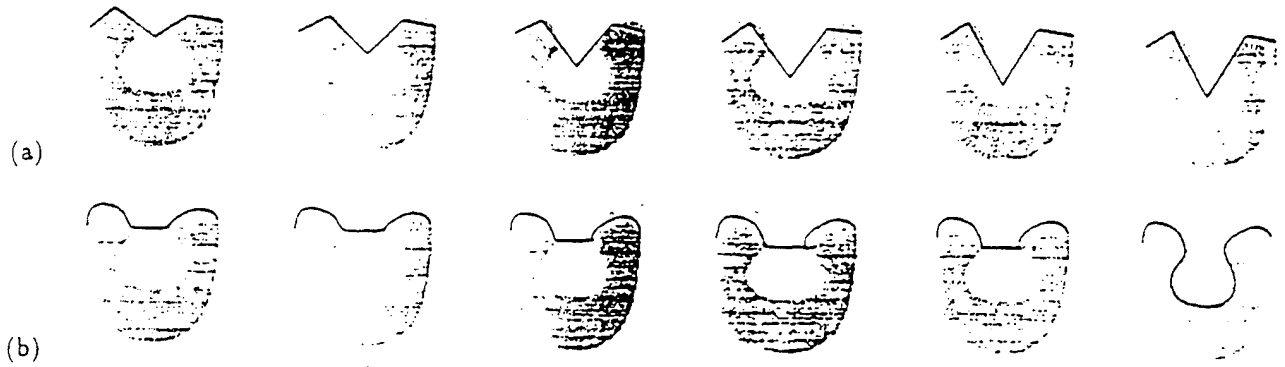


Figure 1: Ill conditioned behavior of the dynamic snakes with respect to initialization . (a) Slightly different initializations: the snake is initialized using a polygon with five vertices. While the third vertex moves closer to the shape, the other vertices are the same for all six situations. (b) The corresponding results: the snake detects the correct contour (6th image pair) when the third vertex is close enough to the object's border. However, it is not intuitively obvious what the threshold is.

2 Dynamic snakes

The original snakes [2] are modeled as time dependent 2-D curves defined parametrically as

$$\vec{v}(s,t) = (x(s,t), y(s,t))_{0 \le s \le 1} , \qquad (1)$$

where s is proportional to the arc length, t the current time and x and y the curve's image coordinates. The snake deforms itself as time progresses so as to minimize an image potential $E_I(\vec{v}) = -\int_0^1 P(\vec{v}(s,t))\,ds$, where $P(\vec{v}(s,t))$ is a function of the image. One typical choice is to take $P(\vec{v}(s,t))$ to be equal to the magnitude of the image gradient. However, because the gradient magnitude can vary rapidly due to contrast changes and to noise, it has proven effective to either clip the derived potential force [4] or replace the gradient magnitude by its logarithm [3]. Alternatively, one can take $P(\vec{v}(s,t))$ to be the Euclidean distance to the closest edge-point in an edge-map computed using an operator such as the Canny edge-detector [9]. The results of all these approaches are very similar. In our implementation, described in section 3, we use the latter where the Euclidean distances are computed using the Danielsson distance transform [10]. Unfortunately, whatever the choice of P, $E_I(\vec{v})$ is typically not a convex functional.

To perform the optimization, following Terzopoulos *et al.*, one must minimize an energy $E(\vec{v})$ that is the sum of $E_I(\vec{v})$ and of a regularization term $E_D(\vec{v})$. Using the thin-plate model, $E_D(\vec{v})$ is taken to be

$$E_D(\vec{v}) = \frac{1}{2}\int_0^1 \alpha(s)\left|\frac{\partial \vec{v}(s,t)}{\partial s}\right|^2 + \beta(s)\left|\frac{\partial^2 \vec{v}(s,t)}{\partial s^2}\right|^2 ds , \quad (2)$$

where $\alpha(s)$ and $\beta(s)$ are arbitrary functions that regulate the curve's tension and rigidity. In the implementation described in section 3, α and β are taken to be constant and are supplied by the user. We have shown [3] that they can be chosen in a fairly image-independent way. Several techniques, however, have been proposed to dynamically adjust the values of α and β along the curve (see [11] for example).

Variational calculus shows that if v minimizes $E = E_D + E_I$ and is sufficiently regular, that is at least $C^4(0,1)$, then it must be a solution of the Euler differential equation

$$-\frac{\partial}{\partial s}\left(\alpha(s)\frac{\partial \vec{v}(s,t)}{\partial s}\right) + \frac{\partial^2}{\partial s^2}\left(\beta(s)\frac{\partial^2 \vec{v}(s,t)}{\partial s^2}\right) = -\nabla P(\vec{v}(s,t)) \qquad (3)$$

Note that, in order to have a unique solution for this equation, one must specify boundary conditions such as the values and derivatives of $\vec{v}(s,t)\big|_{s\in\{0,1\}}$.

In practice, to minimize $E(\vec{v})$, one must discretize the curve $\vec{v}$ by sampling it at regular intervals. We therefore take $\vec{v}$ to be a polygonal curve defined by a set of vertices $\vec{v}^t = (x_i^t, y_i^t)_{0 \le i \le n}$.

Using finite differences, the snake energy $E(\vec{v})$ becomes $E(\vec{v}) = E_I(\vec{v}) + E_D(\vec{v})$ where

$$E_I(\vec{v}^t) = -\sum_i P(x_i^t, y_i^t)$$

$$E_D(\vec{v}^t) = \frac{1}{2}\sum_i \alpha_i\left[(x_i^t - x_{i-1}^t)^2 + (y_i^t - y_{i-1}^t)^2\right] + \frac{1}{2}\sum_i \beta_i\left[(2x_i^t - x_{i-1}^t - x_{i+1}^t)^2 + (2y_i^t - y_{i-1}^t - y_{i+1}^t)^2\right] .$$

Note that $E_D(\vec{v}^t)$ is quadratic and can be rewritten as

$$E_D(\vec{v}^t) = \frac{1}{2}X_t^T K X_t + \frac{1}{2}Y_t^T K Y_t , \qquad (4)$$

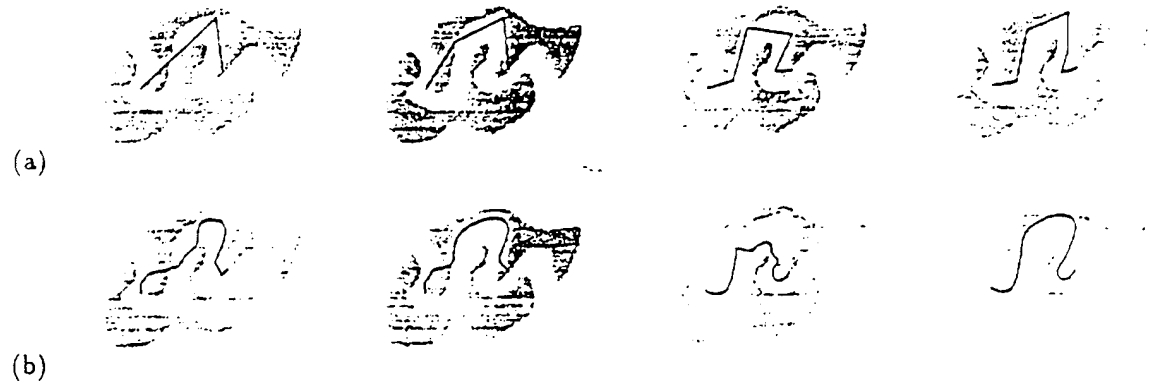


Figure 2: Sensitivity of dynamic snakes to nearby contours. (a) Different initializations: the snake is initialized using a polygon with an increasing number of vertices. (b) The corresponding results: The fact that the two objects are lying close to each other forces the user to outline the desired contour segment precisely. If the snake touches the influence region of a nearby object it will get stuck on the wrong contour.

where K is a $(n+1) \times (n+1)$ penta-diagonal matrix, and $X = (x_0, x_1, \ldots, x_n)$ and $Y = (y_0, y_1, \ldots, y_n)$ are the vectors of the x and y vertex coordinates.
A curve that minimizes the energy E must be such that

$$\frac{\partial E}{\partial \vec{v}} = \frac{\partial E_D}{\partial \vec{v}} + \frac{\partial E_I}{\partial \vec{v}} = 0 \,. \tag{5}$$

Since the deformation energy E_D in equation (4) is quadratic and decouples the x and y coordinates of the curve, equation (5) can be rewritten as a system of two equations

$$KX = F_X \quad , \quad KY = F_Y \tag{6}$$

where

$$F_X = -\frac{\partial E_I}{\partial X} \quad , \quad F_Y = -\frac{\partial E_I}{\partial Y} \,, \tag{7}$$

which are coupled by the "image forces," F_X and F_Y. Note that F_X and F_Y depend on the snake's position, making the system semi-linear. The matrix K, however, is not invertible and these equations cannot be solved directly. This stems from the fact that the Euler differential equation (3) has a unique solution only when boundary conditions are supplied. In section 3, we will show how we can solve this system of equations by supplying the boundary conditions. This will be one of the key differences between our approach and more traditional snake implementations in which the minimization of $E(\vec{v})$ is achieved by embedding the curve into a viscous medium and solving the equation of the dynamics. This amounts to adding a Rayleigh dissipation functional to the energy and leads to solving the following differential equation:

$$\frac{\partial E}{\partial \vec{v}} + \gamma \frac{d\vec{v}}{dt} = 0 \,,$$

where γ is the viscosity of the medium. As in the case of equation (6), after time discretization, this can be rewritten as a set of two equations in X and Y:

$$\begin{aligned} (K + \gamma I) X_t &= \gamma X_{t-1} + F_X \\ (K + \gamma I) Y_t &= \gamma Y_{t-1} + F_Y \end{aligned} \tag{8}$$

where K, F_X and F_Y are defined in equation (6). The matrix $(K + \gamma I)$ is positive definite for $\gamma > 0$. The dynamic snakes are effective when the initial position of $\vec{v}$ is close from the desired solution. However, as illustrated by figures 1 and 2, they are very sensitive to initial conditions. They can easily get caught in local minima when there are other edges in the vicinity of the desired one that may "catch" the snake or when the desired outline presents large concavities that force the snake to extend itself. In the next section, we introduce a different breed of snakes, the "static snakes" that alleviate these problems by replacing the viscosity term by boundary conditions that produce better solutions of equation (3).

3 Static snakes

To improve upon the snakes' convergence properties, we must use constraints that better reflect the image properties than the Rayleigh dissipation functional of section 2. In our implementation, we assume that the user specifies snake end points in the vicinity of clearly visible edge segments, which implies a well defined edge direction. It becomes natural to use these points and their associated edge directions as anchor points and to propagate the edge information along the curve starting from them.

We use these anchor points to derive an initial position for the snake which will, in general, be close to

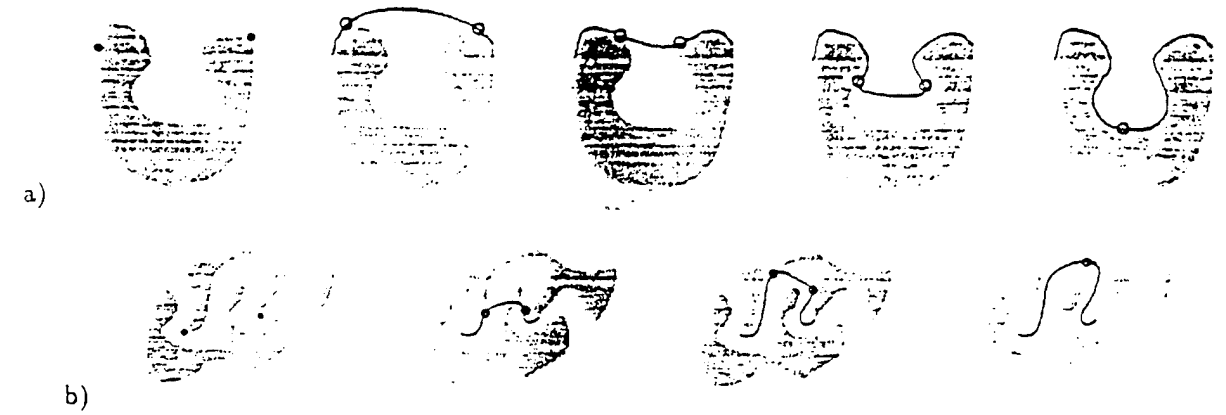


Figure 3: Evolution of a static snake on the synthetic images of figures 1 and 2. The circles denote the farthest vertices away from the end points for which the image forces are turned on. The optimization stops when the two circles meet (section 3.3).

the desired answer in the vicinity of those points but nowhere else. We will therefore "turn on" the image forces (7) in those areas and compute a new position for the snake using the same boundary conditions as before. A longer part of the new solution will be closer to the actual image edge than before; the image forces can then turned on on this longer part and the snake's position recomputed. By iterating this process, we eventually turn on the image forces over the whole length of the snake, thereby achieving the propagation of the edge information from the anchor points to the snake's middle. Our approach is closely related to perturbation theory [12]: we start with an unperturbed solution of our minimization problem and progressively perturb it by considering more and more of the image forces.

In the remainder of this section, we first discuss the use of boundary conditions to solve our minimization problem. We then derive our initial unperturbed snake position from the end points and finally we present the optimization schedule that defines the "turning on" of the image forces.

3.1 Solving the minimization problem with boundary conditions

As discussed in section 2, minimizing the snake's energy amounts to solving the Euler differential equation (3) which leads, after discretization, the semi-linear system of the two equations (6). By fixing the curve's end points (x_0, y_0) and (x_n, y_n) and giving the curve's tangent at those points, the above system of equations reduces to:

$$K'X' = F'_X \quad ; \quad K'Y' = F'_Y$$

where X' is the $n-1$ vector $(x_1, \ldots, x_{n-1})$, Y' the $n-1$ vector $(y_1, \ldots, y_{n-1})$ and K' an $(n-1) \times (n-1)$ penta-diagonal matrix that is now invertible. The system is still semi-linear and cannot, in general, be solved in closed form. Instead, one must still use a time discretization and iteratively solve the system

$$\begin{aligned} K'X'_t &= F'_X|_{X'=X'_{t-1}, Y'=Y'_{t-1}} \\ K'Y'_t &= F'_Y|_{X'=X'_{t-1}, Y'=Y'_{t-1}} \end{aligned}$$

3.2 Initialization

In order to successfully optimize our snake, we must start from an initial position that is approximately correct in the neighborhood of the end points. The easiest way to achieve this result is to solve the homogeneous equations that correspond to the system of Euler equations (3). As discussed in section 2, we take α and β to be constant, and the homogeneous system becomes

$$-\alpha \frac{d^2 v(s)}{ds^2} + \beta \frac{d^4 v(s)}{ds^4} = 0 \qquad (9)$$

where v stands for either x or y and $0 \leq s \leq 1$.

We assume, that the user has chosen the snake's head and tail close to dominant edge fragments. In order to find the true edge location in the near neighborhood of the selected start and end points we first perform a linear search. The snake tangent at its head and tail is then given by the valley direction in the potential surface corresponding to the selected contour fragments. It is obtained from the orientation map that can be computed at the same time as the Canny edge-map. Alternatively, we can fit a cubic polynomial surface to the potential surface in a 5×5 neighborhood of each end point and take the orientation to be the principal direction of minimal curvature [13]. The results are similar. While the tangent direction at the

Figure 4: Outlining roads in an aerial image. (a) Static snakes initializations. (b) Final results. All the road edges are correctly outlined except the bottom left one. (c) The erroneous result is corrected by adding a new control point.

end points can be computed, its orientation cannot be determined. By default, the boundary conditions are chosen so that the initial snake defines acute angles with the line joining the two end-points. Since this heuristic may fail, we provide the user with the possibility to flip the orientation at both ends. This could be automated by initializing the snake in the four possible ways (2 possible orientations at each end) and retaining the best final result.

By construction, this solution has the right tangent at the end points and is close to the right answer near these points. It can therefore serve as an initial curve for the following minimization of the energy functional.

3.3 Optimization procedure

We start the optimization of the energy term by defining the initial snake as the solution of the homogeneous differential system of equation (9). At this stage the snake "feels" absolutely no external potential forces. During the ongoing iterative optimization process the image potential is turned on progressively for all the snake vertices, starting from the extremities. Assuming that the user selects the end points nearby dominant edge fragments in the image, this initialization ensures that the snake lies already close to its optimal position at both ends. We define the "force boundaries" as the location of the vertices farthest away from the end points that feel the image forces. These boundaries approach each other during the ongoing optimization process according to the following rule:

- Each boundary is moved at every iteration step by at least one vertex. To speed up the convergence, we use the fact that our potential surface is the Danielsson distance map. We shift both boundaries across the contiguous vertices whose potential does not exceed the one of the currently active ones by more than 1.

This strategy allows the snake to leave valleys and to close small gaps. However, further investigation is required to better control this "gap closing" mechanism. We are also working on modifying these heuristics to deal with a potential surface derived directly from the image gradients.

4 Results

In this section, using both synthetic and real images, we compare the dynamic snakes with our static snakes and show that the formers' initialization must typically be much closer to the desired answer to achieve comparable results. To achieve a fair comparison, we use the same tension and rigidity parameters, α and β defined in equation (2), for both kinds of snakes.

Figure 3 illustrates a static snakes' behavior on two synthetic images and its ability to outline the cavity and distinguish between two nearby objects. Figure 4 shows that our static snakes can be used to delineate roads in an aerial image using very distant end-points. Note, however, that our snakes can still become confused in the presence of junctions. This is the case for the snake drawn in the bottom left corner of figure 4(b), which is being trapped by an undesirable lo-

cal minimum. In practice, when such problems arise, our interface allows the user to add a new point in the middle of the curve, thereby splitting it into two snakes. Figure 5 shows the snake's performance on an image of an apple.

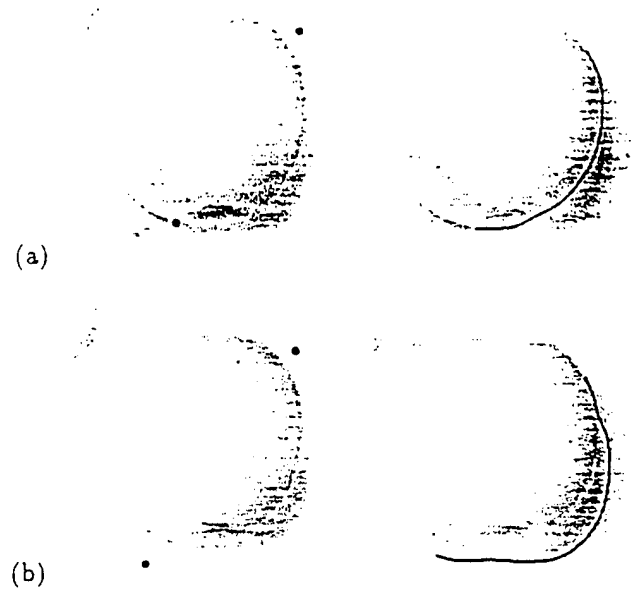


Figure 5: Detection of different image features by static snakes. The apple's outline is detected in (a) whereas (b) shows the detection of the projected shadow.

Nearby features can also be extracted with ease using static snakes. The first row depicts the extraction of the apple's contour. To outline the shadow, as shown in the second row, the initial points simply have to be moved into its vicinity. Figure 6 illustrates similar results on a low contrast face image.

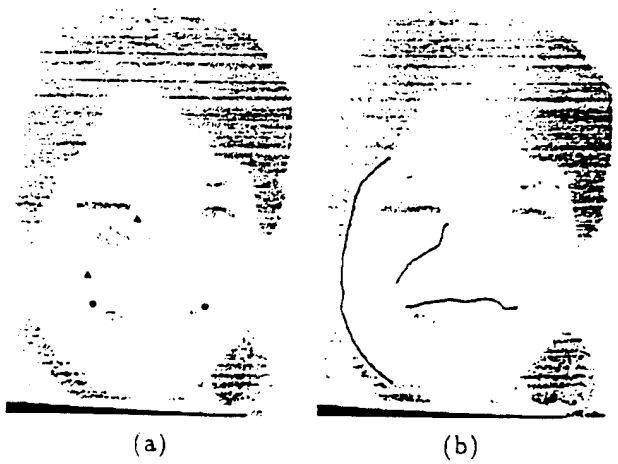


Figure 6: Outlining facial features. (a) Three pairs of end points on a face image (Courtesy of INRIA). (b) Final results.

To segment more complex shapes we need a simple way to sequentially define end points for adjoining open snake fragments. We have implemented an interactive initialization tool in which the contour finding process starts as soon as the user has clicked on two initial points. Sequentially adding salient points will then extend the initial segment under immediate visual control.

Intelligent initialization of snakes is required to make them into an operational tool for image segmentation where existing implementations leave almost all the work to an "expert user" [2]. Our method has been designed for practitioners of image evaluation without a professional background in image understanding.

References

[1] D. Terzopoulos, A. Witkin, and M. Kass. Symmetry-seeking Models for 3D Object Reconstruction. *IJCV*, 1(3):211–221, October 1987.

[2] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *IJCV*,1(4):321-331,1988.

[3] P. Fua and Y.G. Leclerc. Model Driven Edge Detection. *Machine Vision and Applications*, 3:45–56, 1990.

[4] F. Leymarie and M.D. Levine. Tracking deformable objects in the plane using an active contour model. *IEEE PAMI*, 15(6):617–634, 1993.

[5] I. Cohen, L. D. Cohen, and N. Ayache. Using Deformable Surfaces to Segment 3-D Images and Infer Differential Strucutres. *CVGIP: IU*, 56(2):242–263, September 1992.

[6] L.H. Staib and J.S. Duncan. Boundary Finding with Parametrically Deformable Models. *IEEE PAMI*, 14(11):1061–1075, November 1992.

[7] D. Terzopoulos and R. Szeliski. Tracking with Kalman Snakes. In A. Blake and A. Yuille, editors, *Active Vision*. The MIT Press, 1992.

[8] B. Bascle and R. Deriche. Stereo Matching, Reconstruction and Refinement of 3D Curves Using Deformable Contours. In *ICCV*, pages 421–430, Berlin. Germany, 1993.

[9] J. Canny. A computational approach to edge detection. *IEEE PAMI*, 8(6):679–698, 1986.

[10] P.E. Danielsson. Euclidean distance mapping. *CVGIP*, 14:227–248, 1980.

[11] R. Samadani. Changes in connectivity in active contour models. In *Proceedings of the IEEE Workshop on Visual Motion, Irvine, California*, pages 337–343, March 1989.

[12] A. W. Bush. *Perturbation methods for engineers and scientists*. CRC Press, 1992.

[13] R. M. Haralick. The digital step edge from zero crossings of second directional derivatives. *IEEE PAMI*, 6(1):58–68, 1984.

Appendix B

"Using Contextual Information to Set Control Parameters of a Vision Process"

Houzelle, S. and T.M. Strat and P.V. Fua and M.A. Fischler

published in

Proceedings, ICVPR
September, 1994

Using Contextual Information to Set Control Parameters of a Vision Process

S. Houzelle*† T. M. Strat* P. Fua* M. A. Fischler*

* SRI International
Artificial Intelligence Center
333 Ravenswood Avenue
Menlo Park, California 94025
USA

† INRIA
Projet Pastis
BP 93
F-06902 Sophia Antipolis Cedex
FRANCE

ECCV 94 Submission

Abstract

Two of the problems that the user of an image-understanding system must continuously face are the choice of an appropriate algorithm and the setting of its associated parameters. These requirements mean that the user must have a fairly high degree of expertise with the algorithms to accomplish the extraction task effectively. If, on the other hand, the system itself is able to learn how to select among its algorithms and to set their parameters through its experience with similar extraction tasks, it should be possible to reduce the need for operator expertise while improving efficiency at the same time.

A novel approach to supervised learning of image-understanding tactics is based on the notion of contextual information. We use a data base to store past experiences. From this data base, and context elements computed from the task and the input data, we determine whether or not an algorithm is applicable, and which parameters are suitable for it. The data base is regularly updated with information of success or failure of the system. To demonstrate the efficiency of our approach, we describe experiments involving the use of a snake algorithm to perform the task of curvilinear feature extraction. Our implementation allows the various parameters of this technique to be context specific. We show in this setting how our system makes the use of a vision process easier by reducing the needed user expertise and improving efficiency in obtaining the desired results.

Keywords : Image Understanding, Contextual Information, Parameter Learning, Description Learning, Active Contour, Snake

1 Introduction

The research effort described here is an attempt to make computer vision systems more effective by endowing them with a capability to learn. Our philosophy has been shaped by the following principles, each of which has motivated some aspect of our approach:

Learning is essential — image understanding system designers cannot anticipate all possible situations that may arise in advance.

No matter how much effort is devoted to building and refining a knowledge base, there will always be limits to the breadth of competence provided. Even if it were possible to construct a knowledge base sufficient for supporting the entire range of anticipated tasks, the required effort and expense make it infeasible to do so in all but the most limited applications. Effective mechanisms for enabling a system to acquire its expertise over time can have a significant impact on our ability to construct systems that become more (rather than less) competent as they age.

A vision system should improve its performance through experience.

Rather than analyzing images in isolation and throwing away the results, a vision system should interpret an image in the context of what it already knows about the scene. In addition, the results of its interpretation should augment its knowledge of the scene and the extraction task, and the system should be able to use that information to analyze similar situations more effectively in the future.

An intelligent system should never be idle.

If an intelligent system has the ability to learn through experience, it might as well devise its own training examples to more fully exploit that ability. Rather than maintain a static knowledge base when the system is not otherwise engaged, it should concoct new situations or revisit previous ones, invoke its repertoire of reasoning or visual capabilities, and update its knowledge base according to the results.

While the accomplishment of any of these objectives is profoundly difficult, we nevertheless have endeavored to construct a framework that allows the exploration of a particular approach to learning that offers the promise of at least partial solutions. More specifically, our overall goal is to devise a practical computer vision system that can

- Recognize a class of objects in a set of related images
- Build an enhanced description of the environment from the sequence of images it processes
- Use its enhanced description of the environment to improve its recognition capabilities

Imagine a cartographic application in which the photointerpreter's task is to model all roadways within a given geographic area by extracting features from a collection of overlapping overhead images. The photointerpreter is to be assisted by a partially automated system designed to support 3–D map construction.

The traditional interactive approach embodied in today's operational systems [1] can be described as follows:

> The system has a collection of algorithms that are suitable for extracting model instances of certain objects. The user chooses the algorithm to be used to extract a particular object. A menu is provided containing the default values of parameters, which the user can override if he chooses. The algorithm with the designated parameters is then applied to the image(s) producing a resultant model instance. The user then has the option to accept the result, to modify the result manually, to rerun the algorithm with a different set of parameters, or to choose a different algorithm.

Two of the problems that the user of such a system must face are the choice of algorithm and the setting of its associated parameters. These requirements mean that the user must have a fairly high degree of expertise with the algorithms to accomplish the extraction task effectively.

If, on the other hand, the system is itself able to learn how to select among its algorithms and to set their parameters through its experience with similar extraction tasks, it should be possible to reduce the need for operator expertise while improving efficiency at the same time.

The approach we propose and describe in this paper addresses the feature extraction task as follows:

> The system has a collection of algorithms that are suitable for extracting model instances of certain objects. The user chooses the class of objects to be extracted and provides information about his task and the scene being analyzed. The system compares the extraction context to its prior experience with similar extraction tasks, to choose an appropriate algorithm and parameter settings for the current task. The algorithm with the designated parameters is then applied to the image(s), producing a resultant model

[1] e.g., RCDE, GLMX, DSPW, KBVision, Geoset

instance. The user then has the option to accept the result, to modify the result manually, to rerun the algorithm with a different set of parameters, to choose a different algorithm, or to ask the system to provide an alternative selection of algorithm and parameters. The system updates its database of experience with the outcome for use in subsequent extraction tasks.

The focus of this paper is on establishing a foundation for machine learning in interactive image understanding systems — how can a computer vision system use its experience to improve its competence? The design and implementation of a commercially successful interactive computer vision system, the RADIUS Common Development Environment (RCDE), has been discussed elsewhere [HQ88, MWQ+91]; this effort is intended to enhance the performance of RCDE through the addition of a learning component. We build upon the notion of a context-based vision system that has also been previously developed [SF91, Str92].

Our contribution lies in the creation of an architecture that already serves a useful purpose — it successfully learns to select feature extraction algorithms and their parameters. In addition, this architecture has been designed to serve as a foundation for embedding more powerful styles of learning within an interactive vision system. By offering solutions to some of the nonconventional problems in machine learning that arise, we have taken the first steps toward realizing this goal.

The user in our interactive system is a critical component in the automated learning process, even though he is not necessarily aware of this role. The user provides the performance evaluation and correction feedback that is necessary for directed learning and avoids the need for a computationally infeasible trial-and-error approach. Our system takes advantage of the human review of its results to determine how successful it has been, to reinforce its successes, and to take corrective action for its failures.

Our approach is based on the use of contextual information. We define it as any information that may characterize the task or input data given to a vision process. Thus, an image is part of the contextual information, as is the camera geometry, a priori scene knowledge, and the purpose of image analysis. Many authors have used contextual information in image-understanding systems [MHM85, DCB+89, CGHS93, Str93], but few have made the use of context a way to improve the performance of systems through experience and learning.

We have implemented and tested an initial design and demonstrated successful performance. Our experiments involved the extraction of 70 features in four images of two different sites, and encompassed 195 trials. These results are presented in graphical form where the effects of successful learning are clearly apparent.

In the next section, we present in detail our approach to supervised learning. We see what information is stored in the data base and how parameters are retrieved using past experiences. We

also describe the updating process that is critical to continuing improvement of the performance of the system.

In Section 3 and the Appendix, we describe the snake algorithm that we use to demonstrate our system's effectiveness. Snakes [KWT88, TWK87, FL90] are a very powerful technique for edge detection that integrate information from both photometric and geometric models in an optimization framework. More specifically, we show how our implementation allows the various parameters to be context-specific as opposed to image-specific.

Finally, in Section 4, we present some experimental results; we show how our system makes the use of vision algorithms easier by reducing the required user expertise while improving his efficiency.

2 Approach

Let $\mathcal{A}(\mathbf{D}, \mathbf{P})$ be a process that takes two kinds of information as input, data represented by a vector $\mathbf{D}$ (which includes specification of the task), and parameters represented by a vector $\mathbf{P}$. $\mathcal{A}$ gives a solution $\mathcal{S}$ as output. Suppose we have calculated some context information about the task and data represented by a context vector $\mathbf{C}$.

Consider as an example that $\mathcal{A}(\mathbf{D}, \mathbf{P})$ is a segmentation process that starts from the trivial segmentation (one pixel is one region) and merges two regions if the difference between the two grey-level means of the regions is less than a given threshold s. Input data to this particular process is the image I ($\mathbf{D} = (I)$), and $\mathbf{P} = (s)$, the threshold that has to be set depending on the image. One element of context vector $\mathbf{C}$ could be the mean of gray level disparity between two neighboring pixels calculated on every pixel in I.

The basic problem addressed here is to set the values of the parameter vector $\mathbf{P}$ knowing the values of the context vector $\mathbf{C}$. The setting should be optimal in the sense that it gives the expected solution $\mathcal{S}$ for process $\mathcal{A}(\mathbf{D}, \mathbf{P})$. An important consideration is that context and parameter vector elements can be either categorical or numerical, and that one (or more) of the context elements may be unknown.

Note that this problem is not a classic problem of supervised classification where some objects O (parameters) are to be put in one of the classes $\omega_1, ...\omega_n$ (parameter values) knowing some measure vectors $\mathbf{x}$ (context vector). There are two major differences. The first one is that the measure vectors may contain some categorical values or even the value "unknown". The second is that classification generally assumes that there is a fixed number of classes, but here, parameters can be continuous variables[2].

[2]In practice, however, these would be a finite set because of discretization.

This problem is also not a classic problem of interpolation, where we would have to find y (parameter vector) knowing that $y = \mathcal{F}(x)$ (x context vector), where x can be compared to some other context vectors $(x_1, ..., x_m)$ for which we know the associated parameter vectors $(y_1, ..., y_m)$. The main reason is that contextual elements can be categorical and an order relation may not exist between context values. Perhaps more importantly, in both of the above classical problems, the mapping function is known (at least implicitly). In our formulation, a new task can always be introduced, which could alter an existing mapping or even reverse a previous one.

The existence of these two attributes has necessitated the creation of a nonconventional approach to machine learning.

2.1 General Scheme

A primary performance criterion for a practical system for interactive feature extraction is efficiency. Efficiency is the ability to generate a good result in a previously encountered situation while avoiding the repetition of past errors. Thus, it is necessary for the system to keep a record of its successes and failures. The best way to evaluate the system output (and so to determine success or failure) is to ask the user to do it. While this task is generally easy for him, it greatly improves the learning ability of the system.

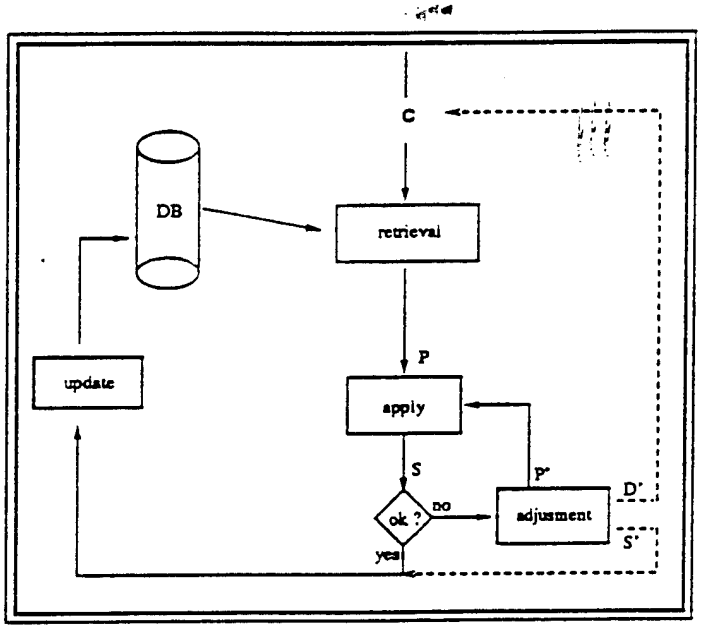


Figure 1: General scheme

From these considerations, we derive the general scheme for our system, depicted in Figure 1. Given a context element vector C, a data base (DB) where past experiences are stored is used to retrieve a parameter vector P. When P is calculated, the process $\mathcal{A}(\mathbf{D},\mathbf{P})$ is applied and the user checks the validity of the result. If the result S is acceptable, P is stored in order to update the parameter value probabilities in the data base. If $\mathcal{A}$ has failed, the user has three adjustment options : the first one is to manually modify the parameters and to apply $\mathcal{A}$ again until it succeeds. The second option is to manually modify the result S so it becomes reasonably good. In these first two cases, C, P, and S are stored to update the data base. This phase is called the learning update, and is usually performed after the session. Finally, the third option for the user is to modify some aspects of the input data D. In such a case, a new context vector is calculated and the procedure starts all over again. In this way, the system is able to improve its performance over time, as the likelihood that the system encounters a context which is similar to one in which it has successfully accomplished its task increases monotonically.

This architecture can serve as the foundation of a practical system for cartographic feature extraction, while affording a path to the creation of a vision system with very powerful and novel constructs for learning. Our current implementation provides limited parameter learning abilities already. More ambitious learning techniques will be installed in the future by replacing the retrieval and update modules in Figure 1 with more capable ones.

In the following sections we describe our current approach to the retrieval of P using the data base, and the update of the data base from new experiences.

2.2 Data Base

The data base is used as a "memory" that contains the past experiences of the system, that is, encountered contexts and associated parameters. Appropriate parameter settings for newly encountered contexts are computed by comparing the current context to previously stored experiences. Particular attention has been paid to defining what information has to be stored, and what the appropriate representation for this information should be.

It is important to note that neither context elements nor parameters can be treated separately from each other — they have to be considered as associated entities. Some parameters are going to be sensitive to more than one context element. Thus, considering context elements separately in the data base is difficult to manage, since they may be associated with some contradictory parameter values or parameter vectors. It is also unreasonable to consider parameters independently (in the data base); they are usually not independent, and this could also lead to inconsistent parameter values. Thus the best solution is to consider each context vector as an entry in the data base, and to associate one (or more) parameter vectors with it.

Another important point is that there is no bijection between a context vector C and a parameter vector P, as there may be several P suitable for a unique C. As an example, consider that the process $\mathcal{A}$ is a car braking system, taking as input a parameter P_1 proportional to the braking force. Suppose we have to adjust this parameter from two context elements, the distance Dt between the car and a traffic light, and the color Co of the traffic light. If $Co = red$, the higher the Dt, the more possibilities there are for P_1 values, depending on whether or not one wants to slow down rapidly. In this example, there is no unique value for P_1, but rather a range of values. This suggests keeping several possible parameter vectors for one context in the data base.

However, it is important to be able to rank these parameter vectors, since some could be spurious. The best parameter vectors (those that have performed successfully in the past) should be tried before any others. Thus, we store the following information in our data base:

- Possible parameter vectors $\mathbf{P}$ for each encountered context element vector $\mathbf{C}$. We call $R(\mathbf{C})$ the set of all $\mathbf{P}$ associated with $\mathbf{C}$ in the data base.
- Two numbers $n^+(\mathbf{P})$ and $n^-(\mathbf{P})$ associated with each $\mathbf{P}$, telling how many times it as been chosen as suitable (n^+), or not (n^-).

$n^+(\mathbf{P})$ and $n^-(\mathbf{P})$ are interesting for the following reasons :

- The set $\{\mathbf{P_0} \in R(\mathbf{C}) \mid n^+(\mathbf{P_0}) = \max_{\mathbf{P} \in R(\mathbf{C})} n^+(\mathbf{P})\}$ allows us to find the best parameter vectors for the given context $\mathbf{C}$.
- $n^+(\mathbf{P})/n^-(\mathbf{P})$ gives an estimate of the reliability of the parameter estimation for the given context $\mathbf{C}$.
- If $n^+(\mathbf{P}) << n^-(\mathbf{P})$, it can be deduced that $\mathbf{P}$ is a spurious vector and that it should be removed from the data base.
- Let $\phi^+(\mathbf{C}) = \sum_{\mathbf{P} \in R(\mathbf{C})} n^+(\mathbf{P})$, and $\phi^-(\mathbf{C}) = \sum_{\mathbf{P} \in R(\mathbf{C})} n^-(\mathbf{P})$.
  If $\phi^+(\mathbf{C}) << \phi^-(\mathbf{C})$, then the expectation that the algorithm succeeds is very low, and it can be deduced that the algorithm is not applicable in the particular context $\mathbf{C}$.

Context elements can be categorical variables as well as continuous variables. Because of storage capacity, every value of a continuous variable context element cannot be stored in the data base. One way to handle this problem is to introduce a discretization step ds. Consider a context vector $(c_{1,i_1}, ..., c_{j,k'}, ..., c_{m,i_m})$ to be introduced in the data base. Suppose that the jth context element is

a continuous variable, and that the data base already contains an entry $(c_{1,i_1}, ..., c_{j,k}, ..., c_{m,i_m})$. If $|c_{j,k'} - c_{j,k}| > ds$ we will consider $(c_{1,i_1}, ..., c_{j,k'}, ..., c_{m,i_m})$ as a new entry in the data base; otherwise, we will update the data base using entry $(c_{1,i_1}, ..., c_{j,k}, ..., c_{m,i_m})$.

If the discretization step is too small, the learning process will be longer, because it will be harder to find the same context vector more than once. The storage capacity will have to be larger too, to store additional cases. Finally, there will be redundancy in the data base, since some different context vectors may have the same set of suitable parameters.

On the other hand, if the discretization step is too high, some parameters associated with one context vector won't be suitable for every case having this context. The number of spurious vectors will be high, and efficiency will suffer.

In the current implementation of the system, the discretization steps have been set manually to what we think are reasonable values. Note that establishing the discretization interval for parameters is much easier because an expert usually knows what the minimum steps are that will impart a difference in the output of the process. Moreover, from previous considerations, we can see that, by checking the degree of redundant parameter vectors and spurious parameter vectors in the data base, discretization steps could be adjusted automatically.

Designing a more competent similarity metric than euclidean distance in context space, and a more effective discretization procedure than fixed distance thresholding, are important problems that we hope to address in the future through learning mechanisms rather than a priori design. For now, our straightforward solutions to these problems allows us to concentrate on other aspects of the design, such as the representation of context elements, the design of the data base, and the accommodation of both numerical and categorical data.

2.3 Parameter Vector Selection

The parameter vector selection is a retrieval process. From a new context vector C, the most suitable parameter vectors for C have to be retrieved from the data base. To do so, the context vector from the data base that is equal to or nearest to the new context vector C has to be found first, and then the best parameters associated with this nearest entry have to be retrieved.

The elements in the context vector C constitute a vocabulary for describing the current extraction context. Ideally, the system would be able to modify or augment this vocabulary with elements that are automatically determined to be critical to the selection of algorithms and parameters. Our current implementation is restricted to a predefined set of context elements.

2.3.1 Finding the nearest context in the data base

A new context vector C is presented to the system. If C is present in the data base, then the retrieval is immediate. In this case and if more than one parameter vector is associated with C in the data base, then the most suitable parameter vector(s) (as defined by $n^+(\mathrm{P})$ and $n^-(\mathrm{P})$) is (are) provided.

The real problem is to deal with context vectors presented to the system when no identical vector is present in the data base. To solve this problem, we look for the nearest context vector present in the data base. This implies finding a similarity measure to compare the new context vector with context vectors of the data base. Many similarity measures can be found in the literature, such as the inner product, the cosine measure, or the dice measure [WWY92]. All these similarity measures assume real valued vectors. Because vectors in the data base are not necessarily real valued, Using a generic similarity measure is questionable because context vectors in the data base are not necessarily real valued and similarity measures for categorical context elements may have to be ad hoc.

A heuristic solution to this problem is to find the nearest context vectors in the sense of the number of equal context values. In this case, the set of context vectors of the data base that have a minimum number of values different from the values of C has to be found. Let us call this set $N(\mathrm{C})$.

Let $V_1, ..., V_p$ be p sets of vectors. We define the **minimal intersection** of $V_1, ..., V_p$ as the function $\bigcap_{min}(V_1, ..., V_p)$ that returns every vector of $V_1, ..., V_p$ that minimize the number of different component values. There may be more than one vector from each set in the minimal intersection.

To better understand the previous definition, consider the following example :

$$\text{Let} \quad \begin{cases} V_1 = ((1\ false\ 0.1\ low),\ (1\ true\ 0.3\ high),\ (1\ maybe\ 0.1\ low)) \\ V_2 = ((1\ true\ 0.1\ low),\ (2\ false\ 0.3\ high)) \end{cases}$$

$$\text{Then} \quad \bigcap_{min}(V_1, V_2) = (((1\ false\ 0.1\ low),\ (1\ maybe\ 0.1\ low)),\ ((1\ true\ 0.1\ low)))$$

The number of differences in the minimal intersection of the previous example is 1 (the second vector element). We call this number the **rank of the minimal intersection.**

Finding elements of $N(\mathrm{C})$ can be achieved using the minimal intersection between C and each context vector of the data base, by selecting vectors that minimize the rank of this minimal intersection.

Let s be the number of entries in the data base, $CS = \{\mathrm{C}\}$ and $CS_i = \{\mathrm{C}_i\}$, $i \leq s$ be the sets containing, respectively, only the new context C and only the ith element of the data base. Let $v_{min,i}$ be the rank of the minimal intersection between CS_i and CS, and $vc_{min} = \min_{j \leq s}(v_{min,j})$; then

$$\forall i \leq s,\ \mathrm{C}_i \subset N(\mathrm{C}) \quad \text{if and only if} \quad v_{min,i} = vc_{min}. \tag{1}$$

Any system that is to base its future actions on prior experiences must have a mechanism for finding matching contexts within its repository of historical information. Since it is unreasonable to expect that the context of every task will have already been encountered, the mechanism must be able to identify experiences that are relevant to the context of the present task. The intermingling of both categorical and numerical context elements has forced us to develop a new mechanism for finding relevant contexts in the database of prior experiences. Equation 1 defines our solution to this nonconventional problem.

2.3.2 Parameter vector retrieval

Since we have found the entries nearest to C in the data base, we now have to find which parameter vectors associated with these nearest entries are the most suitable. Parameter vectors associated with each element of $N(\mathrm{C})$ can be very different, and may not all be acceptable for the new context.

The basic solution is to provide every parameter vector associated with every element of $N(\mathrm{C})$, that is the union of all possible parameter vectors, and to let the user decide which one is the most appropriate.

However, there should be a better solution based on the **intersection** of the selected parameter vectors. If this intersection is not empty, it means that some parameter vectors acceptable for C may also belong to it. Thus, parameter vectors belonging to this intersection can be provided as most suitable for C. If there is no intersection, we can reason the same way in terms of "minimal intersection." In this case, let $PS_1, ..., PS_p$ be the sets of parameter vectors associated with each element of $N(\mathrm{C})$. The minimal intersection of $PS_1, ..., PS_p$ provides a set of potentially acceptable parameter vectors in the new context C.

2.4 Data Base Update

Updating the data base is very important to improving the performance of the system. This process is performed at the end of a session. Thus, running time is not important, while performance is. During the session, update data are accumulated in a file. This file contains all resulting information of a session, that is the success as well as the failure of the system to provide correct parameters. Every attempt to provide a parameter vector is analyzed. If the attempt was a success, that is, if the solution $\mathcal{S}$ returned by the process $\mathcal{A}(\mathbf{D}, \mathbf{P})$ was really the one expected by the user, the automatically retrieved parameter(s), and the manually set parameters (in the case where the user

had to set $\mathbf{P}$ manually), are incorporated into the data base. For each added parameter vector $\mathbf{P}$, the value of $n^+(\mathbf{P})$ or $n^-(\mathbf{P})$ is adjusted.

On the contrary, if the attempt was a failure, that is, if the user had to manually find the solution S that he wanted, then a search process is run to find a set of parameter vectors that best reproduce the given correct solution. The context vector and the best parameter vector(s) are then added to the data base.

Because the data base update is performed off-line, it can be performed continuously whenever the system is not otherwise engaged. We have not yet implemented it, but our design allows for this time to be spent finding new context elements that better resolve the selection of algorithms and parameters. This facility would constitute a very powerful capacity for the discovery of new concepts – a challenging problem in machine learning.

3 Snakes

The automated procedure for parameter setting that we have described is, in theory, suitable for setting the parameters of virtually any algorithm. For purposes of evaluation, we have performed our experimentation using one class of feature extraction algorithms — an optimization approach known as snakes.

Snakes were originated by Terzopoulos, Kass, and Witkin [KWT88, TWK87] and have since given rise to a large body of literature. In the original implementation, the parameters were chosen interactively, and potentially had to be changed from image to image. In our own implementation [FL90], which is further described in the appendix, those parameters are computed automatically and become amenable to context-specific setting.

A 2–D snake is treated as a polygonal curve C defined by a set S containing n equidistant vertices

$$S = \{(x_i\, y_i),\ i = 1, \ldots, n\}$$

that can deform itself to optimize an objective function $\mathcal{E}(C)$.

Formally we can write the energy $\mathcal{E}(C)$ that the snake minimizes as a weighted sum of the form

$$\mathcal{E}(C) = \sum_i \lambda_i \mathcal{E}_i(C)$$

where the magnitudes of the $\mathcal{E}_i$ depend on the specific radiometry and geometry of the particular scene under consideration and are not necessarily commensurate. In order to determine the values of the λ_i weights in a context-specific way as opposed to an image-specific one, we have found it necessary to normalize out those influences. The dynamics of the optimization are controlled by

the gradient of the objective function (Appendix A, Equation 7). We have therefore found that an effective way to achieve this result is to specify a set of normalized weights λ'_i such that

$$\sum_{1 \leq i \leq n} \lambda'_i = 1 \ .$$

The λ'_i define the relative influences of the various components, and we use them to compute the λ_i as follows:

$$\lambda_i = \frac{\lambda'_i}{\| \vec{\nabla} \mathcal{E}_i(\mathcal{S}^0) \|}$$

where $\mathcal{S}^0$ is the estimate at the start of each optimization step. In this way we ensure that the contribution of each $\mathcal{E}_i$ term is roughly proportional to the corresponding λ'_i indepedently of the specific image or curve being considered.

Table 1 lists the parameters of the snake algorithm that we use to test the learning capability of our system. In practice, there are a few more, like those that define the rate of increase of the viscosity or the stopping conditions. However, since the algorithm is not very sensitive to these, we simply fix them once and for all. The categorical parameters determine the type of snake to be used, the presence or absence of a smoothing term, the optimization procedure to be used in the absence of a smoothing term, and whether or not the endpoints of the snake ought to be fixed.

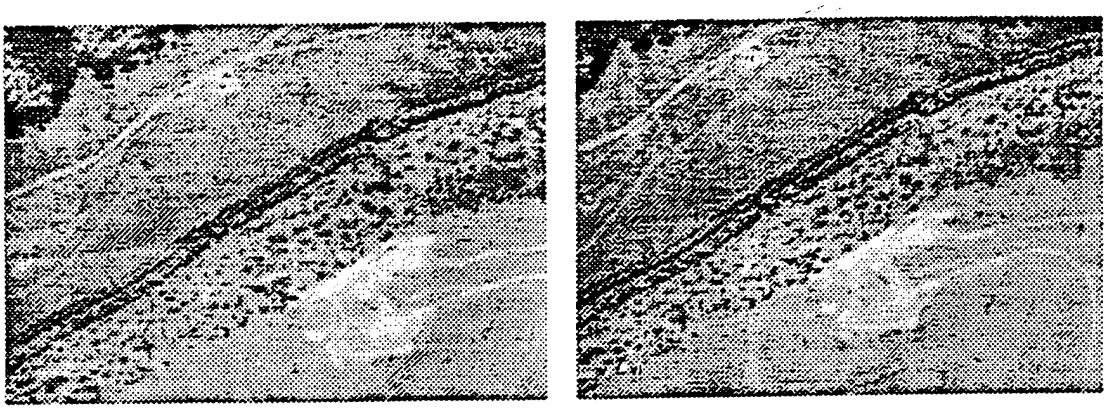
Categorical parameters	
Type of snake	Snakes can model smooth, polygonal or ribbon curves.
Fixed endpoints	The endpoints of the snake can be either fixed or not.
Numerical parameters	
Gaussian smoothing	Size of the gaussian mask used to compute image gradients
Initial step size	Δ_p, pixel step size of Equation 8 used to compute the initial viscosity
Stick length	Initial intervertex spacing of the snake, in pixels
Smoothness constraint	λ'_D weight of the deformation component, Equation 5
Width constraint	λ'_W weight of the width component, Equation 10
Curvature/tension ratio	Relative contribution of tension and curvature, Equations 5 and 9

Table 1: Snake categorical and numerical control parameters. These parameters and related equations are defined in the Appendix.

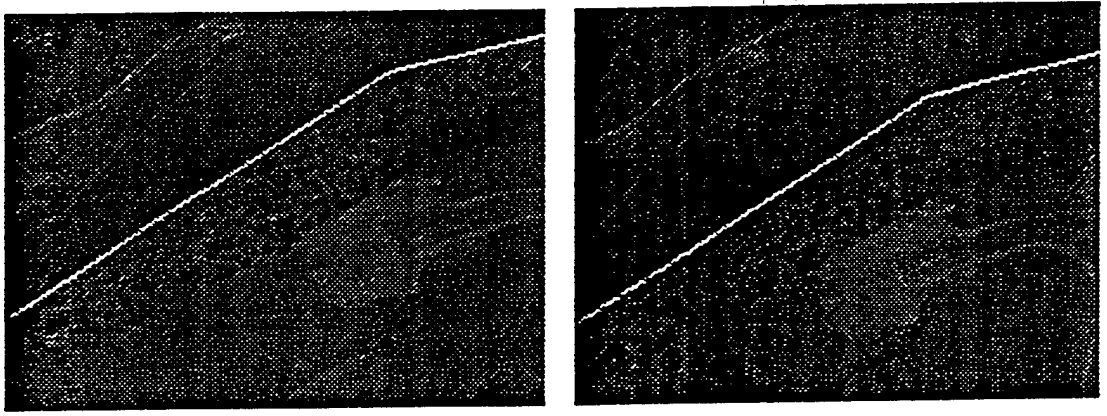
4 Experimental Results: Learning and Selecting Snake Parameters

We have applied our approach to learning the parameters described in Table 1, by implementing the architecture in the RADIUS Common Development Environment (RCDE) [MWQ+91].

First, let us illustrate the general scheme of our system on the following example. Suppose the user's task is to delineate the ridge present in the two images depicted in Figure 2.a. The user sketches the 3–D curve in the left image of Figure 2.b. The camera models and digital terrain model associated with the image site is used to draw the curve in the right image of (Figure 2.b).



(a)



(b)

Figure 2: (a) Two images from one site used in our tests. (b) 3–D seed curve defined in two views

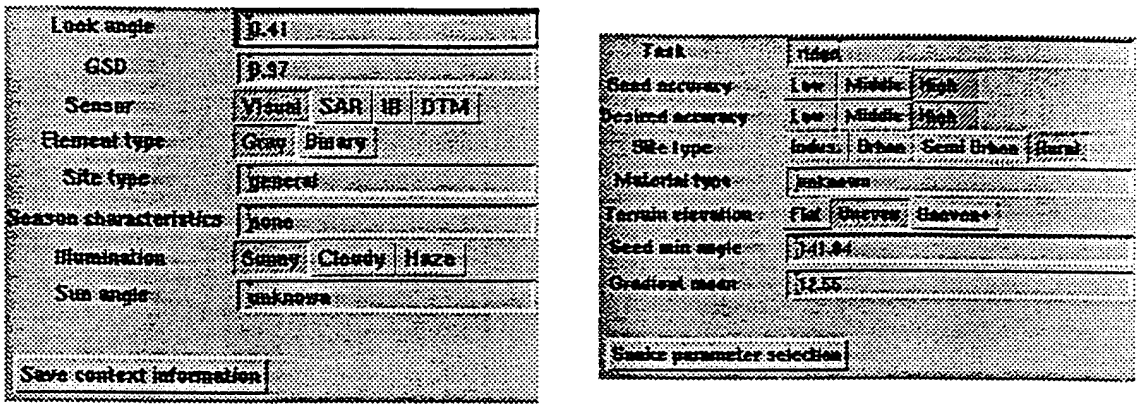


Figure 3: Context menus: global context elements (left), site-specific context elements (right)

Then the user reviews contextual information (Figure 3). There are two menus, corresponding respectively to global image context elements and curve-specific context elements. We have eight global context elements, *Look Angle* giving the look angle of the sensor, *GSD* giving the resolution, *Sensor* which indicates the type of the sensor, *Element type*, *Site type*, *Season Characteristics* which point out some season particularity (snow or rain), *Illumination*, and *Sun Angle* which is important for predicting shadows. We also have eight context elements for characterizing local contextual information : *Task* which indicates the class of object to be extracted, *Seed accuracy* which indicates the distance between the seed curve and the expected solution, *Desired accuracy* which indicates whether or not the user wants the optimized curve to strictly follows the contours of the image, *Site type*, *Material type*, *Terrain elevation*, *Seed min angle* which is the smallest computed angle between two consecutives lines in the seed curve, and *Gradient mean* which is the average of the intensity gradient around the seed curve.

These sixteen context elements form the context vector C. Most of the items are calculated automatically. The user can choose to not provide every item — unknown context elements are ignored. The Parameter Selection button returns a selection of parameters based on the nearest context vector present in the data base. The user can select one of the provided parameter sets and invoke the algorithm. If the user can't find any parameter sets giving an acceptable optimized curve, he can adjust the solution manually, set his own parameters, or modify the initial seed curve (thereby setting a new context). When the user finds an acceptable solution, the initial curve, optimized curve, and parameters are saved to update the data base (note that this is completely transparent to the user). The optimized curve is presented in Figure 4.

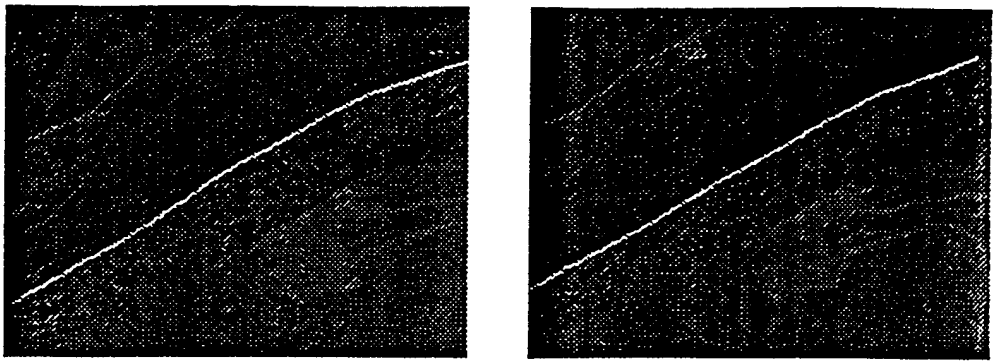


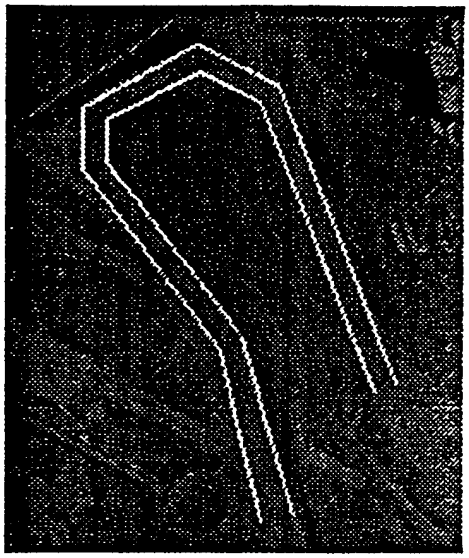
Figure 4: Snake-opimized 3–D curve

Figure 5 and 6 show sub-images of the two sites we used in our tests. A site consists of several images, generally aerial images of dimensions greater than 1000 × 1000 pixels. The two test sites are very different from each other: the first one is a mountainous rural area with several industrial facilities (Figure 5.a), while the second is an urban area in flat terrain (Figure 6.a). Figures 5.b and 6.b show curves used as seeds in the snake optimization process. Figures 5.c and 6.c show the results of the optimization. Although the building boundaries presented in Figures 6.b and c appear very similar, careful inspection will reveal that there are significant differences between the two — the optimized version is much more precise than the sketch. Finally Figures 5.d and 6.d show the parameters provided by the system. The *Smoothness constraint* for the ribbon of the first site is relatively small because of the relatively high curvature of the ribbon. This aspect is captured by the context element *Seed min angle*. *Gaussian smoothing* needs to be smaller for curve of the second site because of the relatively high edge density around the curve, and more particularly, the presence of shadow. This aspect is captured by the *Gradient mean* context element.

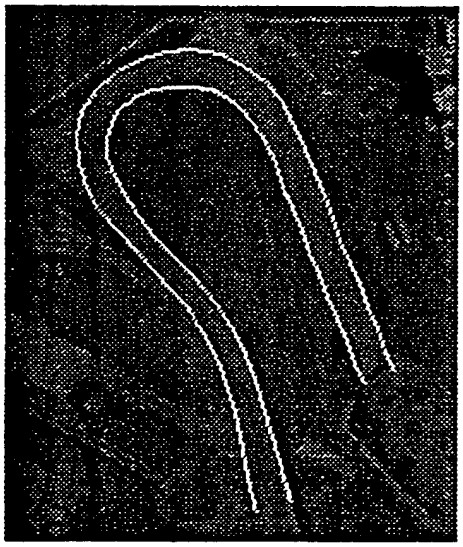
Testing the efficiency of our system with the snake algorithm involves repeating the following steps: define a curve, query the system for appropriate parameters, evaluate the result, and adjust it, if necessary, as described in Section 2.1. As the data base grows, required adjustments should be fewer in number, and the expertise required of the user should decrease.

Two major problems arise with respect to maintaining test objectivity. The first one is the choice of the image curves to optimize and the order in which they are optimized. The best solution would be to define all the curves we want to test, and present them randomly to the user. Because this is not feasible in practice, we adopt the strategy of selecting the curves and processing them sequentially.

(a)



(b)

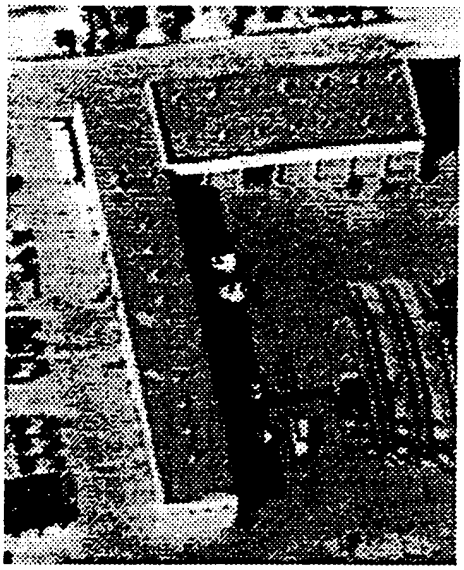


(c)

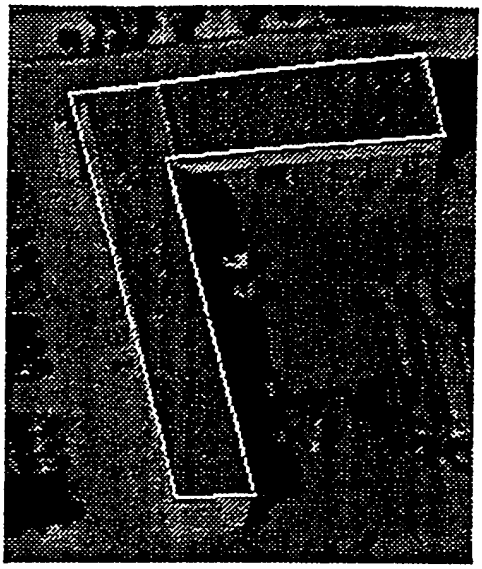
Parameters	Values
Type of snake	ribbon
Fixed endpoints	true
Gaussian smoothing	2
Initial step size	2.0
Stick length	10
Smoothness constraint	0.6
Width constraint	0.5
Curvature/tension ratio	1.0

(d)

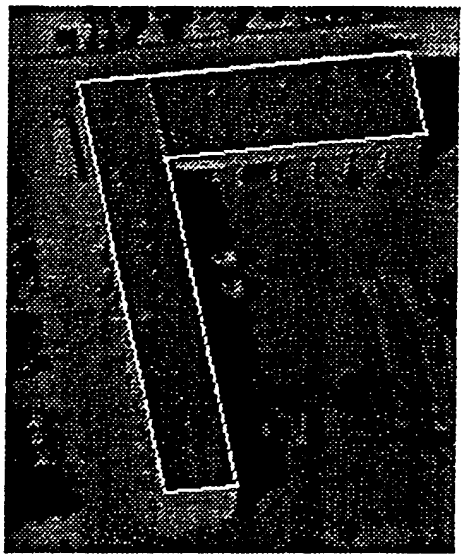
Figure 5: (a) Example of images of the first test site. (b) Ribbon seed curve. (c) Snake-opimized ribbon curve. (d) Provided parameters



(a)



(b)



(c)

Parameters	Values
Type of snake	Polygonal
Fixed endpoints	not used
Gaussian smoothing	1
Initial step size	2.0
Stick length	not used
Smoothness constraint	not used
Width constraint	not used
Curvature/tension ratio	not used

(d)

Figure 6: (a) Example of images of the second test site. (b) Closed 2-D curve. (c) Snake-opimized 2-D curve. (d) Provided parameters

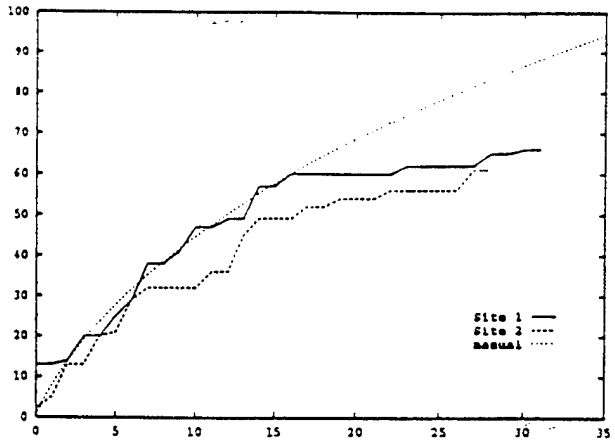


Figure 7: Cumulative number of manual settings of parameters

The second problem is the evaluation of the results. Some results may be acceptable for one user but not for another. The present paper describes an on-going study, and determining how well the learned result will carry over from one user to another has not yet been attempted. All the test results presented below come from a single user.

To show the effectiveness of our system we plot the cumulative number of manual parameter settings as a function of the number of tests (Figure 7). We can see that for both sites, system reactivity is similar. There is a continuous decrease in the slope of each curve. This decrease is due to the effect of successfully learning suitable parameter assignments and represents an improvement in efficiency. The frequency of manual parameter setting that is required clearly decreases and tends toward zero, which is the theoretical ideal. The slope decrease also means that the user needs fewer trials to achieve his goal.

The third curve shows the hypothetical number of manual parameter settings for a user who does not use the learning module, but simply sets the parameters himself before each optimization. This curve fits the two others when the system starts to learn, and then tends to an assymptotic line with slope 2.0, indicating a mean of two manual settings per curve to optimize. The difference between the amount of hypothetical manual parameter setting and the results obtained by a user employing the learning process indicates the improvement in efficiency provided by the learning module. From the graph it is apparent that the improvement increases as the system gains experience with the site.

Without the assistance of the parameter learning module, a novice user can require 10 or more invocations of the snake algorithm before he attains a suitable parameter setting for each seed curve to be optimized. The capacity of the learning module to reduce the required number of invocations

(to less than one per task in our experiments) represents a significant improvement in the efficiency with which snake algorithms can be employed in an interactive system.

5 Conclusion

This paper grew out of an attempt to solve a practical and important problem in interactive scene analysis: the automated selection of feature extraction algorithms and their parameters, as a function of image content and task requirements. An abstract characterization of this problem is that of mapping a multidimensional context space (representing image data and task specification) into a multidimensional algorithm-selection space (the extraction algorithms and their parameter settings). It was immediately apparent that an analytic design was infeasible. Two of the many reasons are:

- We don't have effective ways of analytically describing image content.
- The range of possible tasks and image types is essentially infinite — no a priori design can hope to subsume all possible situations.

A "learning" approach appeared to be the only alternative.

On the other hand, we are not addressing a classical problem in machine learning where some input vector of attribute measurements is mapped into one of a relatively small number of categories. As noted above, the number of distinct points in the algorithm-selection space is (at least in theory) infinite. Further, simple interpolation schemes can't deal with the expected noncontinuous categorical variables. Thus, rather than some form of "parameter learning", we must solve a problem in associative retrieval where the associations must be learned and are subject to change over time.

The fact that the learning system is embedded in an interactive system (to deal with continuous change) offers both a challenge and an opportunity. The human operator must be aided rather than burdened by the presence of the learning system but can provide directed feedback about system performance.

Thus, the primary contribution of this paper lies in the structuring of an interactive, embedded learning system for an important problem in the design of computer vision systems — the automated selection of feature extraction algorithms and their parameters, as a function of image content, collateral data, and task requirements. The framework we have described lays the foundation for new learning mechanisms to be developed and tested — we have taken the first steps toward applying machine learning in a nonconventional learning context. We have also offered solutions to some of the subproblems that arise: how to define similarity of context vectors in which elements are both numerical and categorical; how to choose among the multiple parameter vectors that might be

retrieved from the data base, and how to update the data base with experience gained through continual use of the feature extraction system.

We have implemented and tested an initial design and demonstrated successful performance within a cartographic modeling domain using snake algorithms. Future work remains to establish the utility of this framework to other feature extraction tasks, to clarify the relation between this approach and existing techniques in associative retrieval and machine learning, and to quantify the benefits in a realistic, operational setting.

A Snakes

Here we provide a mathematically precise account of the snake algorithms that we have employed within our system for learning the parameters of vision algorithms.

A.1 2–D Linear Snakes

A 2–D snake is treated as a polygonal curve $\mathcal{C}$ defined by a set S containing n equidistant vertices

$$S = \{(x_i\, y_i),\ i = 1, \ldots, n\} \tag{2}$$

that can deform itself to maximize the average edge strength along the curve $\mathcal{G}(\mathcal{C})$:

$$\mathcal{G}(\mathcal{C}) = \frac{1}{|\mathcal{C}|} \int_0^{|\mathcal{C}|} |\nabla \mathcal{I}(\mathbf{f}(s))|\, ds, \tag{3}$$

where I represents the image gray levels, s is the arc length of $\mathcal{C}$, $\mathbf{f}(s)$ is a vector function mapping the arc length s to points (x, y) in the image, and $|\mathcal{C}|$ is the length of $\mathcal{C}$. In practice, $\mathcal{G}(\mathcal{C})$ is computed by sampling the polygonal segments of the curve at regular intervals, looking up the gradient values $|\nabla \mathcal{I}(\mathbf{f}(s))|$ in precomputed gradient images, and summing them up. The gradient images are computed by gaussian smoothing the original image and taking the x and y derivatives to be finite differences of neighboring pixels. We have shown [FL90] that the points along a curve that maximizes $\mathcal{G}(\mathcal{C})$ are maxima of the gradient in the direction normal to the curve wherever the curvature of the curve is small. Therefore, such a curve approximates edges well except at corners. Unfortunately, $\mathcal{G}(\mathcal{C})$ is not convex functional and to perform the optimization, following Terzopoulos *et al.*, we minimize an energy $\mathcal{E}_(\mathcal{C})$ that is a weighted difference of a regularization term $\mathcal{E}_D(\mathcal{C})$ and of $\mathcal{G}(\mathcal{C})$:

$$\mathcal{E}_(\mathcal{C}) \;=\; \lambda_D \mathcal{E}_D(\mathcal{C}) - \lambda_G \mathcal{G}(\mathcal{C}) \tag{4}$$

$$\begin{aligned}\mathcal{E}_D(C) &= \mu_1 \sum_i (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 \\ &+ \mu_2 \sum_i (2x_i - x_{i-1} - x_{i+1})^2 + (2y_i - y_{i-1} - y_{i+1})^2 \end{aligned} \quad (5)$$

The first term of $\mathcal{E}_D$ approximates the curve's tension and the second term approximates the sum of the square of the curvatures, assuming that the vertices are roughly equidistant. In addition, when starting, as we do, with regularly spaced vertices, this second term tends to maintain that regularity. To perform the optimization we could use the steepest or conjugate gradient, but it would be slow for curves with large numbers of vertices. Instead, it has proven much more effective to embed the curve in a viscous medium and solve the equation of the dynamics

$$\begin{aligned}\frac{\partial \mathcal{E}}{\partial S} + \alpha \frac{dS}{dt} &= 0, \\ \text{with } \frac{\partial \mathcal{E}}{\partial S} &= \frac{\partial \mathcal{E}_D}{\partial S} - \frac{\partial \mathcal{G}}{\partial S},\end{aligned} \quad (6)$$

where $\mathcal{E}$ is the energy of Equation 4, α the viscosity of the medium, and S the state vector of Equation 2 that defines the current position of the curve. Since the deformation energy $\mathcal{E}_D$ in Equation 5 is quadratic, its derivative with respect to S is linear and therefore Equation 6 can be rewritten as

$$\begin{aligned}K_S S_t + \alpha(S_t - S_{t-1}) &= -\left.\frac{\partial \mathcal{E}}{\partial S}\right|_{S_{t-1}} \\ \Rightarrow (K_S + \alpha I) S_t &= \alpha S_{t-1} - \left.\frac{\partial \mathcal{E}}{\partial S}\right|_{S_{t-1}}\end{aligned} \quad (7)$$

where

$$\frac{\partial \mathcal{E}_D}{\partial S} = K_S S,$$

and K_S is a sparse matrix. Note that the derivatives of $\mathcal{E}_D$ with respect to x and y are decoupled so that we can rewrite Equation 7 as a set of two differential equations in the two spatial coordinates

$$\begin{aligned}(K + \alpha I) X_t &= \alpha X_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial X}\right|_{X_{t-1}} \\ (K + \alpha I) Y_t &= \alpha Y_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial Y}\right|_{Y_{t-1}}\end{aligned}$$

where K is a pentadiagonal matrix, and X and Y are the vectors of the x and y vertex coordinates. Because K is pentadiagonal, the solution to this set of equations can be computed efficiently in $O(n)$ time using LU decomposition and backsubstitution. Note that the LU decomposition need be recomputed only when α changes.

In practice α is computed in the following manner. We start with an initial step size Δ_p, expressed in pixels, and use the following formula to compute the viscosity:

$$\alpha = \frac{\sqrt{2n}}{\Delta_p}\left|\frac{\partial \mathcal{E}}{\partial S}\right|, \tag{8}$$

where n is the number of vertices. This ensures that the initial displacement of each vertex is on the average of magnitude Δ_p. Because of the non linear term, we must verify that the energy has decreased from one iteration to the next. If, instead, the energy has increased, the curve is reset to its previous position, the step size is decreased, and the viscosity recomputed accordingly. This is repeated until the step size becomes less than some threshold value. In most cases, because of the presence of the linear term that propagates constraints along the whole curve in one iteration, it takes only a small number of iterations to optimize the initial curve.

The snakes described above have proved very effective at modeling smooth curves. Some objects, however, such as buildings, are best modeled as polygons with sharp corners. They can be handled in this context by completely turning off the smoothness term. Such objects typically have a relatively small number of corners, and the optimization is performed using a standard optimization technique.

A.2 3–D Linear Snakes

Snakes can be naturally extented to three dimensions by redefining $\mathcal{C}$ as a 3–D curve with n equidistant vertices $S = \{(x_i\ y_i\ z_i)\},\ i = 1, \ldots, n\}$ and considering its projections in a number of images for which we have accurate camera models. The average edge strength $\mathcal{G}(\mathcal{C})$ of Equation 3 becomes the sum of the average edge strengths along the projection of the curve in the images under consideration, and the regularization term of Equation 5 becomes

$$\begin{aligned} \mathcal{E}_D(\mathcal{C}) &= \mu_1 \sum_i (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 + (z_i - z_{i-1})^2 \\ &+ \mu_2 \sum_i (2x_i - x_{i-1} - x_{i+1})^2 + (2y_i - y_{i-1} - y_{i+1})^2 + (2z_i - z_{i-1} - z_{i+1})^2 \end{aligned} \tag{9}$$

Since the derivatives of $\mathcal{E}_D$ with respect to x, y, and z are still decoupled, we can rewrite Equation 7 as a set of three differential equations in the three spatial coordinates:

$$\begin{aligned} (K + \alpha I)X_t &= \alpha X_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial X}\right|_{X_{t-1}} \\ (K + \alpha I)Y_t &= \alpha Y_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial Y}\right|_{Y_{t-1}} \\ (K + \alpha I)Z_t &= \alpha Z_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial Z}\right|_{Z_{t-1}} \end{aligned}$$

where X,Y, and Z are the vectors of the x,y, and z vertex coordinates.

The only major difference with the 2–D case is the use of the images' camera models. In practice, $\mathcal{G}(C)$ is computed by summing gradient values along the line segments linking the vertices' projections. These projections, and their derivatives, are computed from the state vector S using the camera models. Similarly, to compute the viscosity, we use the camera models to translate the average initial step Δ_p, a number of pixels, into a step Δ_w expressed in world units and use the latter in Equation 8.

A.3 Ribbons

2–D snakes can also be extended to describe ribbon-like objects such as roads in aerial images. A ribbon snake is implemented as a polygonal curve forming the center of the road. Associated with each vertex i of this curve is a width w_i that defines the two curves that are the candidate road boundaries. The state vector S becomes the vector $S = \{(x_i\ y_i\ w_i)\},\ i = 1,\ldots,n\}$ and the average edge strength the sum of the edge strengths along the two boundary curves. Since the width of roads tends to vary gradually, we add an additional energy term of the form

$$\begin{aligned} \mathcal{E}_W(C) &= \sum_i (w_i - w_{i-1})^2 \qquad (10) \\ \Rightarrow \frac{\partial \mathcal{E}_W}{\partial W} &= LW, \end{aligned}$$

where W is the vector of the vertices' widths and L a tridiagonal matrix. The total energy can then be written as

$$\mathcal{E}(C) = \lambda_D \mathcal{E}_D(C) + \lambda_W \mathcal{E}_W(C) - \lambda_G \mathcal{G}(C)$$

and at each iteration the system must solve the three differential equations:

$$\begin{aligned} (K + \alpha I)X_t &= \alpha X_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial X}\right|_{X_{t-1}} \\ (K + \alpha I)Y_t &= \alpha Y_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial Y}\right|_{Y_{t-1}} \\ (K + \alpha I)W_t &= \alpha W_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial W}\right|_{W_{t-1}} \end{aligned}$$

2–D ribbons can be turned into 3–D ones in exactly the same way 2–D snakes are turned into 3–D ones. The state vector S becomes the vector $S = \{(x_i\ y_i\ z_i\ w_i)\},\ i = 1,\ldots,n\}$ and at each iteration the system must solve four differential equations, one for each coordinate.

References

[CGHS93] V. Clement, G. Giraudon, S. Houzelle, and F. Sandakly. Interpretation of remotely sensed images in a context of multi sensor fusion using a multi-specialist architecture. *IEEE Trans. on Geoscience and Remote Sensing*, 1993.

[DCB+89] B.A. Draper, R.T. Collins, J. Brolio, A. R. Hanson, and E.M. Riseman. The schema system. *International Journal of Computer Vision*, 3(2):209–250, 1989.

[FL90] P. Fua and Y.G. Leclerc. Model driven edge detection. *Machine Vision and Applications*, 3:45–56, 1990.

[HQ88] A. J. Hanson and L. Quam. Overview of the sri cartographic modeling environment. In *DARPA Workshop on Image Understanding*, pages 576–582, April 1988.

[KWT88] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

[MHM85] D. M. McKeown, W. A. Harvey, and J. McDermott. Rule-based interpretation of aerial imagery. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 7(5):570–585, September 1985.

[MWQ+91] J.L. Mundy, R. Welty, L. Quam, T. Strat, W. Bremmer, M. Horwedel, D. Hackett, and A. Hoogs. The radius common development environment. In *Proc. of AIPR, Washington, DC*, October 1991. (also *in Proc. of DARPA Image Understanding Workshop*, San Diego, California, 1992.).

[SF91] T. M. Strat and M. A. Fischler. Context-based vision: Recognizing objects using both 2d and 3d imagery. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 13(10):1050–1065, October 1991.

[Str92] Thomas M. Strat. *Natural Object Recognition.* Springer-Verlag, New York, 1992.

[Str93] Thomas M. Strat. Employing contextual information in computer vision. In *DARPA Workshop on Image Understanding*, 1993.

[TWK87] D. Terzopoulos, A. Witkin, and M. Kass. Symmetry-seeking models and 3D object reconstruction. *International Journal of Computer Vision*, 1:211–221, 1987.

[WWY92] Z.W. Wang, S.K.M. Wong, and Y.Y. Yao. An analysis of vector space models based on computational geometry. In *ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 152–160, June 1992.